FM-7／NEW7／77
マシン語入門マニュアル
中村英都
SHUWA SYSTEM TRADING CO.,LTD.

FM-7（セブン）/NEW7（ニューセブン）/77（セブンセブン）
マシン語入門マニュアル

中村 英都

SHUWA SYSTEM TRADING CO., LTD.

目次

第０章 なぜマシン語か ——前書きにかえて——

基礎編

第１章 マシン語学習の前に ——予備知識と概念——

第２章 マシン語とモニタ ——モニタの使い方——

第３章 マシン語プログラミング part Ⅰ ——レジスタと基本命令——

第４章 マシン語プログラミング part Ⅱ ——加算と減算——

第5章 マシン語プログラミング part III ——比較とループ——

第6章 マシン語プログラミング part IV ——サブルーチンとスタック——

第7章 マシン語プログラミング part V ——アドレッシングモード——

第8章 マシン語プログラミング part VI ——論理演算と残りの命令——

応用編

第9章 FM-7シリーズの構成の環境 ——マシン語の観点から——

第$A章 プログラミング -アセンブラの使い方を中心に-

第$B章 BIOS ——入出力の基礎——

第$C章 サブシステム ——グラフィックを使う——

第 $ D 章
プログラミング実践
―ゲームプログラムを題材に―

第 $ E 章
プログラミング探究
―サブCPUのコントロール―

第 $ F 章
BASICとマシン語
―BASICとの連携―

付　　録

第 0 章

なぜマシン語か

――前書きにかえて――

富士通のパーソナルコンピュータＦＭ―7は、その発売（1982年11月）以来、数多くのマニアに支持され、着実にその販売台数を伸してきました。そして今回、ＦＭ―7とのソフト完全互換性をうたったＦＭ―NEW 7、ＦＭ―77を新たなラインナップに加えて、ますますユーザの裾野を広げていこうとしています。

これらＦＭ―7シリーズ（本書では、ＦＭ―7／ＦＭ―NEW 7／ＦＭ―77を総称してこう呼んでいます）は、その頭脳に『究極の8 bitマイクロプロセッサ』と称される６８Ｂ０９を搭載しており、その点で大きな可能性を秘めています。しかし、ＦＭ―7シリーズはその多くが各人のプログラムの作成においては、ＢＡＳＩＣ専用マシンとして使われており、本書で取りあげるマシン語は、若干影に隠れている感があります。一方、市販のソフトなどをみるとどうでしょうか。簡単な金種計算や速度を要求しない用途のプログラムを除けば、良いソフトとしてユーザに認められているもののほとんどは、マシン語によるプログラムであることに気づかれると思います。

個人がちょっとした用途にプログラムを作成するというときに要求されるのは、その実行速度というより、デバッグ（虫とり。プログラムの間違いを直すこと）のしやすさという点でしょう。この点で、ＢＡＳＩＣは会話型のプログラミングが可能ですから，重宝な存在です。マシン語は、これとは反対にデバッグには若干手間がかかるものの、実行速度に関しては、ＢＡＳＩＣなど足もとにもおよびません。例えば、マシン語でプログラムを作成する際、多少冗長にプログラムを組んだとしても、ＢＡＳＩＣより遅くなるなどということはまずありません。実行速度だけではありません。ＢＡＳＩＣなどの高級言語ではその図体が大きいために、小回りがきかないことがあります。しかし、マシン語はその名のとおり『マシン』に直結した言語ですから、小回りがきくことはもちろん、『マシン』の秘めた全能力を引き出すことが可能です。市販の良いソフトがマシン語で組まれているという理由もこのあたりにあります。また、マシンの全能力を引き出せるからこそ、マシン語

を学ぶ意味があるわけです。

しかし、一方でマシン語学習において、問題となる点がないわけではありません。マシン語というのは、ＢＡＳＩＣなどに比べてより機械に近い言語ですから、おのずと人間の側からすれば複雑にならざるをえません。例をあげれば、ＢＡＳＩＣでは数行で書くことのできるプログラムが数十行に及ぶということもめずらしくありません。マシン語の学習にはこの点からくる，難しさと面倒くささが伴うのはやむをえません。しかしマシン語を学んでいこうという意欲さえあれば、このハードルは容易に越えることができます。

それでは、マシン語を操れるようになるためには、どのように学習を進めていくのがよいのでしょうか。

まず第１に、**マシン語という言語自身を学習し理解**しなければいけません。つまり各命令の動作を理解するわけです。また同時に、これと密接な関係のある（一般的な）コンピュータの構造や、２進法などについても学習する必要があります。これにより、マシン語がわかるというレベルにまで達することができます。しかし、それだけではマシン語のプログラムは組めません。

そこで第２に、**マシン語プログラムを組む方法**について学習しなければいけません。つまり、マシン語の各命令を、有機的に組み合わせてプログラムを作成する方法を学ぶわけです。たとえれば、単語を継ぎ合わせることによって、文章を作成するようなものです。ここまでくれば、マシン語のプログラムが組めるという段階まではいくでしょう。しかし実際に、ＦＭ―7シリーズの上でプログラムを走らせたりして、マシン語を使いこなすには、これだけの学習では不充分です。

第３に求められるのが、**ＦＭ―7シリーズのマシン自体に対する理解**です。既に述べたように、マシン語はマシンに直結した言語です。ですからＢＡＳＩＣのようにハードウェアの違いをある程度吸収して、文字列の表示はＰＲＩＮＴ文でというようにはいきません。つまり、ハードウェアが違えば（つまりマシンが違えば、ということです。日立のＳ１とＦＭ―7シリーズなどのような違いをさしています。ＦＭ―7シリーズの間では、同じと考えてさしつかえありません）画面への表示や、周辺機器の操作方法も違ってきます。だからこそ、使用するマシン自体に対して十分に理解しないと、マシン語を十分に使いこなすということはできません。

以上の点を、踏まえて、本書はＦＭ―7シリーズ上での**マシン語の基礎から応用・実践まで**を目的として以下の構成を取っています。

まず『基礎編（第１章～第８章）』では、マシン語の各命令を重要な順かつ理解しやすい順に解説し、マシン語プログラミングの基礎を伝授します。つまり基礎編で、「マシン語が理解できる」というところまでいくことになります。

次に『実践編（第９章～第＄Ｆ章）』では、基礎編での学習を活かして、「FM―7シリーズ上でのマシン語プログラミング」という点を重視し、マシン語プログラムの組み方と、FM―7シリーズのマシン自身に関する知識を習得することになります。内容としては、ＢＩＯＳやサブシステム（どちらもFM―7シリーズでマシン語プログラムを作成する際には、必ず知っておかなければならない事項です）の使い方をマスターし、最後には、サブＣＰＵ（本文中で詳しく解説します）を直接制御する方法までを学習、高速なリアルタイムゲームの制作に挑戦し、FM―7シリーズの真髄に迫ります。また、ＢＡＳＩＣとマシン語の組みあわせ方にも言及します。

このように、内容は盛りだくさんですが、基礎から積み重ねていけば、中・上級者であるあなたにも、また全くの初心者であっても、必ず最後まで到達できます。そして本書がFM―7シリーズのマシン語を含めたより深い理解につながることを希望します。

最後に、本書の執筆･出版にあたって、多くの方々にお世話になりましたことを、深く感謝します。

1984年６月

中村英都

基礎編

第 1 章

マシン語学習の前に

——予備知識と概念——

1. パソコンの基本構造

コンピュータは、基本的に大型コンピュータであっても、パソコンであってもその構造に違いはありません。コンピュータは図1—1に示すように、おおよそ4つの部分から成り立っています。

入出力装置とは、コンピュータに情報を入力したり、反対に取り出したりする装置で、人間とコンピュータの情報交換はすべてこの入出力装置をとおして行われます。これには、キーボードやCRT*1、プリンタなどが含まれます。

演算装置は、コンピュータ内の情報に演算をほどこして、情報を加工するところです。

記憶装置は、メモリ(memory)とも呼ばれ、入力された情報や加工された情報を記憶して保存しておく場所です。

そして**制御装置**は、上記の各装置の動作をコントロールする役目があり、いわばコンピュータの“指揮者”ともいえます。マシン語を理解して各命令の動作の指揮をとるのも、この制御装置です。

では、実際のFM—7シリーズで詳しく見ていきましょう。入出力装置のうち入力装置の代表はキーボードです。一方、出力装置はFM—7シリーズのシステム全体（CRTやプリンタを含めた全部）で考えればCRTとプリンタです。しかしFM—7シリーズ本体だけでみればこれらを内蔵しているわけではありません（事実CRTその他は外づけです)。ではどういうことかというと、FM—7シリーズ本体には、CRTやプリンタなどの出力装置へ情報を“橋渡し”する回路（これを**インタフェース回路**といいます）を内蔵しているのです。このインタフェース回路は、コンピュータが出力する情報を、出力装置が利用できる形の情報に変換して出力装置へ送る役割りを持っています。

記憶装置はFM—7シリーズ内部に**主記憶装置**（メモリ）があり、記憶される情報には、各種のデータの他にプログラムも含まれます。さらにFM—7シリーズには、**補助記憶装置**としてカセットテープレコーダやフロッピ・ディスク装置など

をつなぐことができ、補助的ではあるものの大量の情報を記憶（記録といった方がいいかもしれません）させることができます。

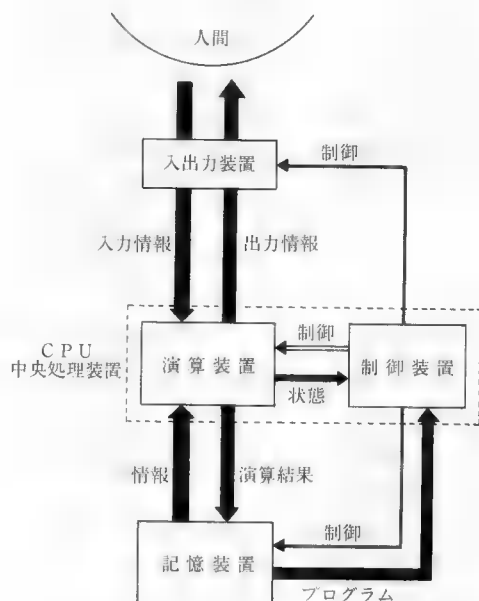



図1－1　コンピュータの基本構造

次に演算装置へいきたいところですが、実はFM－7などのパソコンでは演算装置と制御装置が1つのＬＳＩ*2（大きさは100円ライターぐらいです）に収められています。この演算装置と制御装置をあわせたものをＣＰＵ*3と呼び、FM－7シリーズには6809（ロクハチマルキュー）*4と呼ばれるＣＰＵが搭載されています。このＣＰＵには8080（ハチマルハチマル）、Ｚ80（ゼットハチマル）、6502（ロクゴーマルニー）など数多くの種類がありますが、それぞれ異ったマシン語を持っており、互換性*5は一部の組合せを除いて全くありません。すなわち、これから学ぼうとしているマシン語はあくまで6809用のマシン語であり、Ｚ80用のマシン語ではありません。例えばＺ80をＣＰＵとして搭載しているＰＣ8801などには、6809のマシン語は通じません。この点に十分注意してください。しかし6809のマシン語がマスターできた後ならば、Ｚ80のマシン語は非常に簡単に理解できます。FM－7シリーズにはＺ80カードを取り付けることができるので、この本をマスターしたらＺ80のマシン語をかじってみるのもよいかもしれません。

このようにFM－7シリーズは、4つの装置が有機的に結合されて、すばらしいコンピュータを構成しているわけです。

＊1）ＣＲＴ：Cathode Ray Tube（陰極線管）の略、ディスプレイ、モニタなどとも呼ばれています。
＊2）ＬＳＩ：Large Scale Integration（大規模集積回路）の略。トランジスタや抵抗などの素子を1つの小片の上に数万個配置して1つの機能をなすものです。ＬＳＩやＩＣ（Integrated Circuit：集積回路）などを俗に「チップ」「石」などと呼ぶこともあります。
＊3）ＣＰＵ：Central Processing Unit（中央処理装置）の略。68系（ロクハチケイ）（6800や6809などのＣＰＵの仲間）ではＭＰＵ（Micro～）ということもありますが、意味は同じでパソコンの頭脳です。「系」という表現が出ましたが、他に80系（ハチマルケイ）（8080やＺ80など）や65系（ロクゴーケイ）（6502など）などがあります。
＊4）6809：ＣＰＵの種類を示す番号です。FM－7シリーズには、（建前上）ＭＢＬ68Ｂ09が搭載されています。ＭＢＬは富士通、Ｂは最高サイクル周波数2ＭＨzということを示している記号ですので、マシン語には関係ありません。とにかく名称に68と09という文字が含まれていれば、マシン語はすべて同じです。ちなみに私のFM－7にはＨＤ 68Ｂ09が載っていました。
＊5）互換性：コンパティビリティ（compatibility）ともいいます。お互いに、わずかな変換または全くの変換なしでソフトウェアが交換できることをいいます。俗に「コンパチ」ともいいます。例えば「FM－7とFM－8はＢＡＳＩＣレベルでコンパチだ」などといいます。

2. *2進数とビット*

前節でさかんに情報という言葉を使いました。一般に情報はいろいろな形（例えば文字や絵、音など）で表現されていますが、コンピュータは2進数で表された情報を扱うことができます。2進数とは0と1だけで数を表現するもので、0から1、10、11、100、101、……というように、2ごとに位をあげていく数です。ですから位どりは右（下位）から1（$=2^0$）の位、2（$=2^1$）の位、4（=

2^2)の位、8（＝2^3)の位、16の位、32の位、……ということになります。図1—2に10進数との変換方法を図示しました。この図では4桁の2進数までしか対象にしていませんが、4桁の変換ができれば十分です。

このような2進数をコンピュータは扱うわけですが、この2進数の1桁のことをビット(bit：binary digit）と呼びます。ですからビットとは、コンピュータにおける情報の最小単位であるわけです。

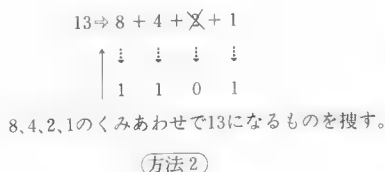
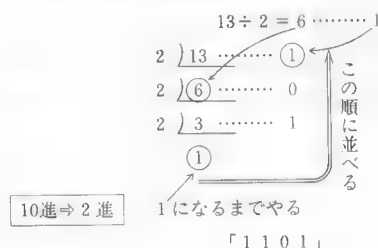
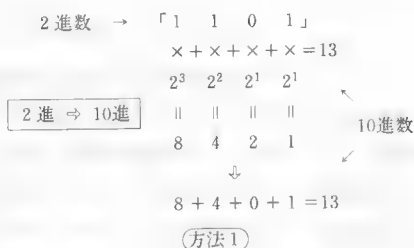



図1－2　2進 ⇔ 10進変換

3. 16進数とバイト

コンピュータに情報を与えるのに、いちいち2進数で「01011111」などと書いていたのでは、手間がかかりすぎます。そこで2進数を右（下位）から4ビットずつに区切って、その4ビットの情報（16種類）を1文字で表そうというのが16進数です。4ビットの情報を1文字で表すには16種類の記号（数字）が必要です。そこで10進数の0から9までの10種の記号にアルファベットのA、B、C、D、E、F、の6種の文字を用います。そして図1—3のように対応させることにします。

2進数	16進数	10進数
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

図1—3　2↔16↔10進対応表

このように対応させると16進によって2進数の良さを失わずに数の桁数を圧縮することができます。16進数では16ごとに位を上げるので、位どりは右（下位)から1の位、16の位、256の位、4096の位というようになります。

いろいろな進数が出てきましたが、ただ「10」と書いたのでは何進数であるかわかりません。そこで本書ではこれから、

10進数 ➡ 18d　　or　18
2進数 ➡ 10010b　or　%10010
16進数 ➡ 12h　　or　$12

という表記をとります*1。

18＝%10010＝$12

です。

```
2進数   0 1 0 1 1 1 1 1 b
        └──┘└──┘
16進数     5       Fh
           ↓       ↓
        5×16¹ + 15 = 95
```

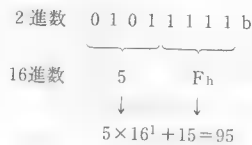



図1－4　2進→16進

ところで人間は細かな情報はひとまとめにして、ひとまわり大きな情報として扱った方が便利なことを知っています。コンピュータも人間の作った創造物だけあって例外ではありません。FM－7シリーズのＣＰＵ6809は、ビットという細かな情報を8つずつ“たば”にして一度に扱うように作られています*2。この8つのビット(つまり8ビット）をひとまとめにした単位をバイト（byte）と呼びます。6809は8ビットを単位として処理するのでバイトは非常に重要な単位といえます。1バイト＝8ビット＝4ビット×2ですから1バイトは2桁の16進数で表されることになります。ですから、1バイトで表現できるデータの数値は＄0～＄ＦＦまでの256種類となります。

最後に1つだけ注意しておきます。10進数とか16進数を扱ってきましたが、これはすべて人間側にわかりやすくするためのものであって、あくまでコンピュータ内部では2進数が使われています。2進数というよりも、1と0の列といった方が正確でしょう。なぜなら同じ1バイトの「１０１０１１１１ｂ」でもこれが数値を表しているのか文字なのか、はたまた絵の一部であるのかは、プログラムを作った人にしかわからないからです。

ちなみにＢＡＳＩＣでは、10進数の他に、16進数と8進数が扱えます（この“扱える”というのは、16進数の文字を内部で2進数の数値に変換する機能をもっているということです。たとえばＢＡＳＩＣであっても内部では2進数を用いており、入出力の際に2進数の数値を10進数の文字（例）に変換しているので、10進数を扱えるのです)。図1―5にＢＡＳＩＣで書いた10進↔16進変換のプログラムをあげておきます。

＊1）“ｄ”はdecimal、“ｂ”はbinary、“ｈ”はhexa-

〔図1－5〕

```
10 '
20 ' Hexadecimal <=> Decimal Convert
30 '
40 INPUT "Hex($) or Dec number ";N$……10進ならその数を，16進なら頭に“$”をつけて入力する
50 IF LEFT$(N$,1)<>"$" THEN V=VAL(N$):GOTO 70 ……………10進のときの処理
60 V=VAL("&H"+MID$(N$,2)) ……………………………16進のときの処理
70 PRINT CHR$(30);TAB(20); ………………………1行上の20ケタ目にカーソルを移動
80 PRINT USING"##### = $@";V;RIGHT$("000"+HEX$(V),4) ……………10進と16進を表示
90 GOTO 40……………………………………………………………繰り返し

RUN
                         10 = $000A
                        100 = $0064
                       1000 = $03E8
                      10000 = $2710
                         16 = $0010
                        256 = $0100
                       4096 = $1000
                      65535 = $FFFF
Hex($) or Dec number ?

Break In 40
Ready
```

decimalの頭文字です。

＊2）FM—7シリーズが「8ビットのパソコン」、CPU6809が「8ビットCPU」と呼ばれる理由です。

4. メモリとアドレス

この章の第1節で、コンピュータにはメモリというものがあり、情報を記憶することができると述べました。それでは6809には、どれだけ情報の記憶場所を持つ機能があるのでしょうか。6809と限らず「8ビットCPU」は、一度に65536個*1の記憶場所を持つことができます。ところでメモリにはプログラムやデータが格納されているわけですが、CPUは数多くの記憶場所をひとつひとつ、区別しなければなりません。このためCPUはメモリに0から＄FFFF（＝65535）までの番号を割り当てて扱っています。この番号のことを**アドレス**（address：番地）といいます。このようにして区別されたひとつひとつのメモリには1バイトの情報を記憶させることができます。1つのメモリに1バイトですから全体では、65536×8ビット、そして1バイトの各ビットには、図に示すように、ビット0からビット7までの名前がついています。

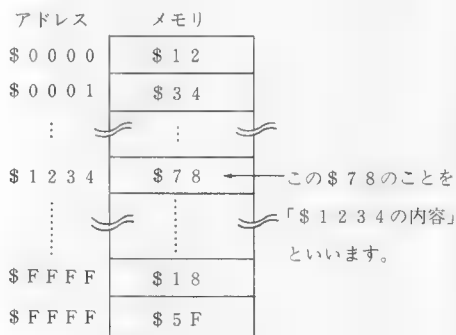



図1−6　メモリのイメージ図

アドレスは＄0000から＄FFFFとつけられていますが、いい換えると、16ビットの数でメモリを区別していることになります。実際CPUからは、16本のアドレスバスと呼ばれる線が出ていて、この上の16ビットの数で、メモリの1つを指定しています。さらに、指定されたメモリの内容は、読出しの場合、データバスと呼ばれる線の上を通って1バイトの情報としてCPUに受け取られます。一方、書込みの場合には、CPUから1バイトの情報がデータバスを通って、(アドレスバスで指定された）メモリに書き込まれるというしくみになっています。

＊1：「メモリ64Kバイト実装」などというのは「65536バイト実装」ということです。つまり1K＝2^{10}＝1024です。このK（大文字で書くのが普通）はK（キロ）と読むことが多いのですが、kg（キログラム）のk（キロ）（こちらは小文字）と区別するためにK（ケイ）と読む場合もあります。

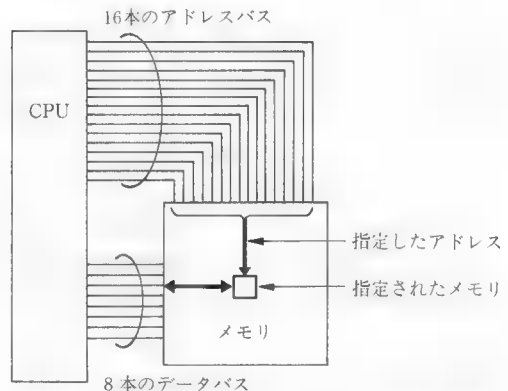



図1−7　アドレスバス・データバス

5. マシン語とBASIC

前節まででマシン語を学ぶための最低限の必須用語は解説しました。この節では少し話題を換えてマシン語の世界からみたBASICについて考えてみます。

第1節で、マシン語を理解するのは（CPUの中の）制御装置であるといいました。ここで理解してほしいのは、CPUはマシン語しか理解でき

ないということです。FM—7に限らずパソコンは本質的にはマシン語しか理解できないのです。

「嘘をいうな！BASIC言語を理解するじゃないか」という人があるかもしれません。電源を入れるとすぐにBASIC言語が使える状態になる今のパソコンでは、これは仕方のないことかもしれませんが、少しずつ種明しをしていきましょう。

実はBASIC本体は、マシン語でできているのです。CPUは動き始める（電源を入れる）と、マシン語でできているBASIC本体を単なるマシン語のプログラムとして実行します。

このマシン語プログラムは、'F－BASIC Ver.3.0…'などと表示し、'Ready'と出力してキーボードからの入力を待つように作られており、CPUはそのとおりに動作します。言葉が混乱しそうなので新しい言葉を定義しましょう。(BASIC）インタプリタ——上の説明で“マシン語でできているBASIC本体”といったのがこれです。その意味についてはこれから明らかにしていきますが、とにかく「大きなマシン語のプログラム」であることを押えておいてください。

話を元に戻して、その後のCPUの動作をみてみます。もしあなたがここで'10　INPUT　A↵'と入力したとすると、CPUはBASICインタプリタの手順にそってこの行をメモリに格納します。続いて'20　B＝A＊2↵'、'30　PRINT“2バイ＝”；B↵'と入力すれば、CPUは順々にメモリにこのBASICプログラムを格納します。ですからこの時点でメモリには今入力した3行のBASICプログラムが格納されていることになります。しかしCPUはマシン語しか理解できないですから、メモリ中のBASICプログラムをじかに実行することはできません。

次に'RUN↵'と入力すると、CPUはBASICインタプリタを実行しながら、入力が「BASICプログラムを実行せよ」と言っていると解釈し、メモリ上のBASICプログラムを実行します。では実行はどのように行われるのでしょうか。その一部を追ってみましょう。

CPUはメモリ上のBASICプログラムから'INPUT'という文字の列をインタプリタの手順にそって解釈し、インプット文であることを知り、インタプリタ内の入力を促す(マシン語の)サブルーチンを呼び出して入力を得ます。得られた入力値を変数A用として確保したメモリに格納して行番号10の処理を終了します。次にCPUは行番号20の処理を開始します……。

このように、インタプリタは各種の文に対応した処理サブルーチンを内蔵したプログラムであり、それはすべてマシン語で書かれ、CPUがこれを実行することにより、人間からみればBASICを理解するコンピュータができあがるというわけです（わかりましたか？わからなくても先へ進んでけっこうです。またいつか読み直してください)。

第２章

マシン語とモニタ

――モニタの使い方――

1. マシン語の姿

既に数多く「マシン語」という言葉を使ってきましたが、ここでその意味を再確認してみましょう。「マシン語（機械語：machine language）」を定義するとすれば、機械（マシン＝ＣＰＵ）が直接解読して実行できる言語となります。すなわち、ＣＰＵは記憶装置に格納されているマシン語プログラムを順々に取り出し、解読後実行していくことになります。その姿はどうかといえば、ＣＰＵが解読するのに容易であるように、以下の形をしています（16進を示す“＄”は省略）。

８Ｅ　８７　０２　ＢＤ　９Ｂ　ＤＢ　７Ｅ　ＡＢ　Ｆ４[*1]

すなわち１バイトの数の並びというわけです。このような１バイトの数を、マシン語そのものの姿であることから「マシンコード」といいます。次

アドレス	マシンコード
＄５０００	＄８Ｅ
＄５００１	＄８７
＄５００２	＄０２
＄５００３	＄ＢＤ
＄５００４	＄９Ｂ
＄５００５	＄ＤＢ
＄５００６	＄７Ｅ
＄５００７	＄ＡＢ
＄５００８	＄Ｆ４

①

```
＄５０００　　＄８Ｅ　＄８７　＄０２　＄ＢＤ
＄５００４　　＄９Ｂ　＄ＤＢ　＄７Ｅ　＄ＡＢ
＄５００８　　＄Ｆ４
```

②

```
５０００　　８Ｅ　８７　０２　ＢＤ
５００４　　９Ｂ　ＤＢ　７Ｅ　ＡＢ
５００８　　Ｆ４
```

③

図2－1　マシン語の姿

に考えなければならないのは、このマシンコードの格納場所です。ＢＡＳＩＣに行番号があったように、マシンコードにも、アドレスをつけて表現しなければ、その格納場所がわかりません。ですからマシン語は図2—1①のようになります。しかし、最初の＄８Ｅがどこに格納されるかがわかれば、それ以降のマシンコードは順々に格納されるのですから、アドレスは自明となります。そこで図2—1②のように表現することもできます。さらに、これらの数は16進であるのも自明なので、“＄”は多くの場合省略され、図2—1③のようになります。

これらの表現は既に雑誌などでみたことがあるのではないのでしょうか。これからは、このマシンコードがどういう動作をする命令なのかを理解していくことになります。

＊1）このプログラムは実際に動作するものです。この章ではこれを用いて「モニタ」と呼ばれるものを学習していくことにします。

2. マシン語モニタ

ここでは、前節で例にあげたマシン語プログラムを実行するにはどうしたらよいのかを順に説明していきます。まずFM—７シリーズの電源を入れると、図2—2のように表示され入力待ちとなります。ところが、これはＢＡＳＩＣのコマンドレベルであるので、マシン語を入力することはできません。そのため、FM—７シリーズは「モニタ」と呼ばれるマシン語の入力・実行を行うプログラムを内蔵しています。そこで、'MON⏎'と入力してください。これにより、「モニタ」が起動しモニタが“＊”を出力してモニタの入力待ちになります（図2—3）。

これでモニタに制御を移すことができました。このモニタには４つのサブコマンドがあり次の機能を持っています（図2—4）。

◀Mコマンド▶

メモリにマシンコードを入力するのに利用するサブコマンドです。“M”に続けてアドレスを入力して“⏎”を入力すると、そのアドレスとそのアドレスのメモリの内容が表示されて入力待ちになります。

内容を変更するときは、16進数を入力して

〔2−3　モニタの起動〕

```
Ready ……………………… ＢＡＳＩＣが“Ready”
MON  ………………………… モニタの起動

*■ ………………………………… カーソル点滅
 ……………………………………… モニタがReady
```

コマンド名	機　　能
M	メモリの内容を変更します
G	指定アドレスに分岐します
R	レジスタの内容を表示し，変更を可能にします
D	指定アドレスから64バイトの内容を表示します

（F−ＢＡＳＩＣ文法書３−３５より）

図２−４　モニタのサブコマンド

〔図２−２ F−ＢＡＳＩＣの起動〕

```
DISK VERSION
How many disk drives      ?  }
How many disk files(0-15)?   } ……………… ＤＩＳＫ　ＢＡＳＩＣのときのみ出力

FUJITSU F-BASIC Version 3.0
Copyright (C) 1981 By FUJITSU/MICROSOFT
25775 Bytes Free
 …………………………………………………………………… 場合によって異なる
Ready
■ ………………………………………………………………… カーソル点滅
```

"↵"、変更しないときはただ"↵"を入力します。そうするとアドレスが1つ進んで再び入力待ちになります。入力をMコマンドは終了するときには、16進数以外の文字（たとえば"."（ピリオド））を入力して"↵"とするとブザーを鳴らして終了となります。

"M"のあとのアドレスは省略することもでき、その時には直前に使用されたアドレスが使われます。図2—1のプログラムを、入力例として入力していったものを以下に示します（図2—5, 6)。

〔図2－5　Mコマンド(1)〕

〔図2－6　Mコマンド(2)〕

◀Dコマンド▶

入力したマシンコードが正しくメモリに格納されているかどうかを調べるのには、Dコマンドを用います。DコマンドはMコマンドと同様に"D"に続けてアドレスを入力し"↵"とします。すると、アドレスに続いて1行に8バイトずつメモリの内容が出力されるのがわかります（図2—7)。

図2—7のようにマシン語を出力したものを、ダンプリストともいいます。さらに続けてみたいときには、アドレスを省略して"D↵"と入力することにより＄５０４０からの64バイトを表示させる

〔図2－7　Dコマンド〕

```
*D5000↵ ……Dコマンドの実行
                                        ……入力したマシンコード
5000 8E 87 02 BD 9B DB 7E AB
5008 F4 00 00 00 00 00 00 00
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00     ……メモリの内容
5028 00 00 00 00 00 00 00 00       （入力された内容の確認）
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■ ……アドレス
```

〔図2－8　Gコマンド〕

```
                                        ……実行開始アドレス
*G5000↵ ……Gコマンドの実行
FUJITSU F-BASIC Version 3.0
Copyright (C) 1981 By FUJITSU/MICROSOFT

*■……実行が終了して再度コマンド待ちになっている
```

ことができます。

もし入力間違いがあったときは再度Mコマンドで修正します。

◀Gコマンド▶

それではいよいよプログラムを実行してみましょう。"G"に続けて実行を開始するアドレスを入力して"⏎"を押します（図2—8)。

いつも見慣れたメッセージが出力され、再度モニタのコマンド待ちになっています。つまり図2—1のプログラムはこのメッセージを出力するプログラムだったということです。

◀Rコマンド▶

残りはこのRコマンドだけですが、このサブコマンドの解説はもう少し後で行います。というのは、Rコマンドを理解するにはレジスタを知らなければならないからです。

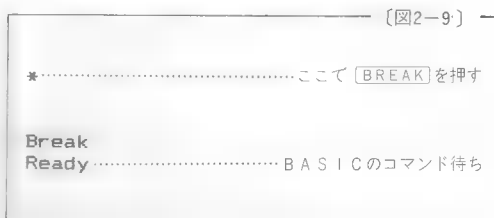
3. マシン語のセーブとロード

前節でマシン語の入力の方法がわかりました。でも電源を切ると、入力したものは一瞬のうちに消えてなくなってしまいます。ＢＡＳＩＣのようにプログラムをテープやディスクに保存する方法を考えましょう。まずはモニタから脱出します。

◀モニタ入力待ちからの脱出▶

モニタから脱出するには"ＢＲＥＡＫ"キーを押します。コントロールキーを押しながらＣまたはＸ（コントロールＣ、コントロールＸ）を押しても同じように脱出できます。

〔図2—9〕

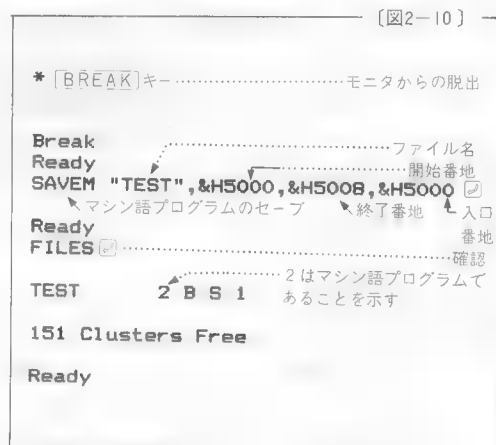
```
*...................ここで BREAK を押す

Break
Ready ..............ＢＡＳＩＣのコマンド待ち
```

◀ＳＡＶＥＭ▶

マシン語プログラムをテープやディスクにセーブするのにはＳＡＶＥＭコマンドを用います。これはＢＡＳＩＣのコマンドです。シンタックス(文法）は次のようになっています。

ＳＡＶＥＭ"ファイルディスクリプタ", 開始番地, 終了番地, 入口番地

ここでいう開始番地、入口番地とはそれぞれ、プログラムの先頭番地、実行を開始する番地を示します。前節のプログラムを例にとれば、図2－10のようになります。

〔図2－10〕

```
* BREAK キー ..................モニタからの脱出

Break                ..............ファイル名
Ready                ..............開始番地
SAVEM "TEST",&H5000,&H5008,&H5000 ⏎
  マシン語プログラムのセーブ   終了番地  入口番地
Ready
FILES⏎ ..............................確認
TEST        2 B S 1   2はマシン語プログラムであることを示す
151 Clusters Free
Ready
```

◀ＬＯＡＤＭ▶

これもＢＡＳＩＣのコマンドですが保存されたマシン語プログラムをロードする際は、ＬＯＡＤＭコマンドを用います。シンタックスは次のとおりです。

ＬＯＡＤＭ"ファイルディスクリプタ"[,［オフセット値][, R]]　　[ ] 内は省略可

通常はファイルディスクリプタのみで図2－11のようにとするとロードできます。

〔図2－11〕

```
LOADM "TEST"⏎ ..............マシン語プログラムのロード
Ready
```

もし何かの都合で、セーブしたときと違ったアドレスにロードしたい場合には、オフセット値を設定します。オフセットを省略すると、セーブ時と同じアドレスにロードされます（図2―12)。

〔図2－12〕

```
LOADM "TEST",&H1000

Ready
MON⏎
*D6000⏎……  $5000←セーブした時の先頭番地
               +
             $1000←オフセット
6000 BE 87 02 BD 9B DB 7E AB
6008 F4 00 00 00 00 00 00 00
6010 00 00 00 00 00 00 00 00
6018 00 00 00 00 00 00 00 00
6020 00 00 00 00 00 00 00 00
6028 00 00 00 00 00 00 00 00
6030 00 00 00 00 00 00 00 00
6038 00 00 00 00 00 00 00 00
*
```

さらに、ロード後すぐにマシン語プログラムを実行したいときは、“，R”をつけて図2―13のようにすることによって、Gコマンドを実行する手間が省けます。

以上の SAVEM LOADM は BASIC の命令ですのでBASICのコマンド待ちで入力するか、BASICのプログラム中でなければなりません。

既にマシン語プログラムを実行させる方法として“Gサブコマンド”をあげました。これはモニタのコマンドですから、BASICのコマンド待ちからマシン語プログラムを実行させるのには、'MON⏎'でモニタに移ってからGコマンドを実行しなければなりません。しかしこれでは不便なのでBASICには次の命令が備わっています。

◀EXEC▶

これはBASICから直接マシン語プログラムを実行させるための命令です。シンタックスは次のとおりです。

EXEC［開始番地］

ここでいう開始番地とは、実行を開始するアドレスのことでSAVEMの開始番地とは異なります。この開始番地を省略すると、直前に実行されたLOADMコマンドの入口番地(実行開始番地)、または直前に実行されたEXECコマンドの開始番地となります。

以上で、マシン語プログラムの入力、実行、保存ができるようになりました。

〔図2－13〕

```
                    ……カンマ(,)が2つあることに注意(オフセット値を省略)
                    ……ロード後に即実行することを示す
LOADM "TEST",,R⏎
FUJITSU F-BASIC Version 3.0
Copyright (C) 1981 By FUJITSU/MICROSOFT  ……実行した結果

*■
```

〔図2－14〕

```
LOADM "TEST"⏎ ……マシン語プログラムのロード

Ready ……ロード終了
EXEC &H5000⏎ ……プログラムの実行
FUJITSU F-BASIC Version 3.0
Copyright (C) 1981 By FUJITSU/MICROSOFT  ……実行結果

*
```

4. マシン語入力ルーツの紹介

さて、前節ではFM―7シリーズ内蔵のモニタを使った入力方法を述べてきましたが、いざ、長いマシン語のプログラムを入力するとなると不便なことに気付くでしょう。特にMコマンドで毎回⏎キーを押さなければならないというのは不便きわまりありません。

そこでマシン語の入力をより簡単にするために『ＭＩＴ７（Machine code Input Tool)』なるプログラムを作成しました。慣れれば、このプログラムを使用することによって１Ｋバイト（1024バイト）を30分で入力することもできるようになります。そうすれば、雑誌などのマシン語プログラムの入力が容易になり、ひいてはマシン語の理解につながる（？）かもしれません。

◀ＭＩＴ７の入力▶

まずFM―７シリーズのモニタによって図2―20のマシン語プログラムを入力して下さい。そして次に決して実行せずにＳＡＶＥＭでテープがディスクにセーブして下さい(ＳＡＶＥＭ“ＭＩＴ７”，＆Ｈ１０００，＆Ｈ１４３７，＆Ｈ１０００)。

次に図2―18のＢＡＳＩＣプログラムを走らせて

　ＳＴＡＲＴ　ＡＤＤＲＥＳＳ＝＄１０００⏎

　ＥＮＤ　　　ＡＤＤＲＥＳＳ＝＄１４３７⏎

として図2―20の一番右の２桁の数字が一致しているか確認してください。この数字は、チェックサム（Check sum）と呼ばれるもので、例えば＄１０１０番地からの１行（16バイト）を加えたものの下２桁が＄１８となっています。もし、入力間違いがあったとすると16バイトの合計が合わなくなり、間違いがすぐに見つけられます。ただし、このチェックだけでは完璧ではないのでその点に注意してください（図2―15参照)。

研究熱心なあなたのために付録にソースリストをつけました。この本を理解した段階で解読してみるのもよいでしょう。

◀ＭＩＴ７の使い方▶

図2―16でまずセーブされたＭＩＴ７を①でロードしたのち、②で実行させます。すると画面が消去されて、左下に③と表示されるので、マシン語入力をする先頭アドレスを入力してください。すると画面いっぱいに図2―15のような行が出力され、入力待ちとなります。ここで16進数を入力するとカーソルの位置に入力されます。このときＡからＦのかわりに、図2―17のように10キーを活用することもできます。またカーソル移動キーによって任意の位置にカーソルを移動できます。さらに１番上（または下）の行にカーソルがあるときにカーソルを上（下）に移動させようとすると画面が下（上）にスクロールします。つまり＄００００から＄ＦＦＦＦまでのメモリを巻絵のようにして、その上を画面が移動するような感覚でマシン語を覗き、入力することができるわけです。この他のコマンドについては次にあげます。

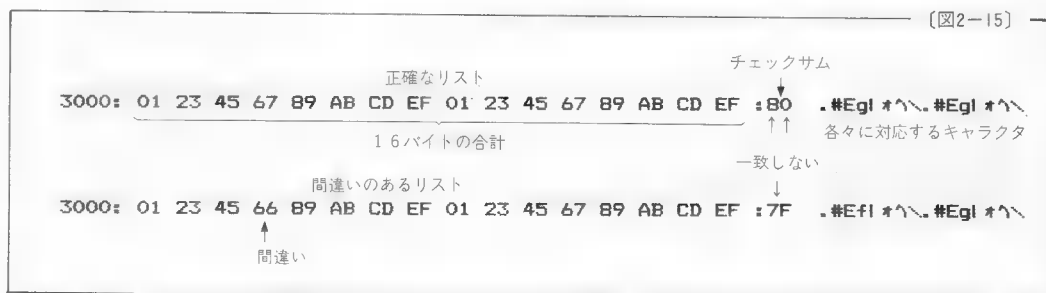
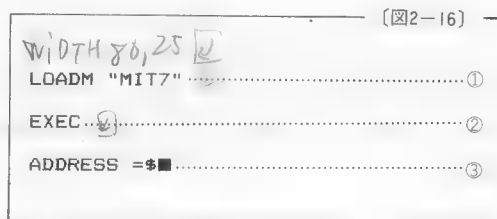
〔図2―16〕

```
LOADM "MIT7" ..................................①
EXEC ⏎ ........................................②
ADDRESS =$■ ...................................③
```

〔図2―15〕

```
                         正確なリスト                              チェックサム
                                                                      ↓
3000: 01 23 45 67 89 AB CD EF 01 23 45 67 89 AB CD EF :80   .#Egl ｵﾍ＼.#Egl ｵﾍ＼
                                                          ↑↑    各々に対応するキャラクタ
                       16バイトの合計
                                                        一致しない
                   間違いのあるリスト                       ↓
3000: 01 23 45 66 89 AB CD EF 01 23 45 67 89 AB CD EF :7F   .#Efl ｵﾍ＼.#Egl ｵﾍ＼
               ↑
              間違い
```

SAVEM "MIT7", &H0850, &H0C9D, &H085D

A面 001 ～ 006

MIT7のコマンド

[TAB]：アドレス入力を行う（Teach Address Base）。

[ESC]：BASICに戻る（ESCape MIT7）。

[DUP]：プリンタへ出力する(DUmp to Printer)
プリンタへ出力する先頭アドレスと終了アドレスを入力する。

↲, ⇐：カーソルを左へ移動する。

・(ピリオド)：カーソルを右へ移動する。

A	B	C	D
7	8	9	E
4	5	6	F
1	2	3	↲
0		・	

↲ はカーソルを左に
・ はカーソルを右に
動かす

図2－17

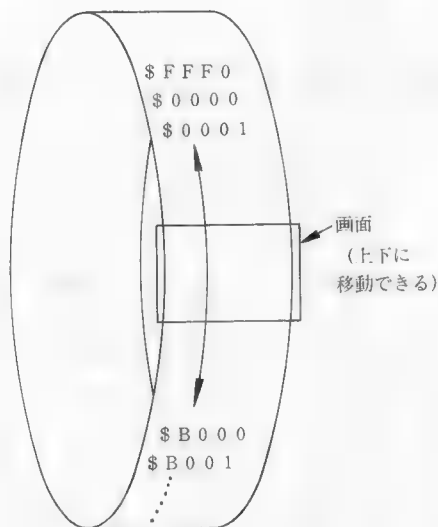



図2－18　MIT7の画面

〔図2－18〕

```
100 ' DUMP & OUTPUT CHECK SUM
110 CLEAR,&H1FFF ································· $2000番地以降を保護
120 INPUT "START ADDRESS =$",A$:S=VAL("&H"+A$) ⎫
130 INPUT "END   ADDRESS =$",A$:E=VAL("&H"+A$) ⎭···· ダンプするアドレスの入力
140 FOR I=S TO E STEP 16
150    T=0
160    PRINT RIGHT$("000"+HEX$(I),4);":"; ··············· アドレスを出力
170    FOR J=0 TO 15
180       D=PEEK(I+J) ·································· データの読み出し
190       T=T+D ······································· チェックサムに足す
200       PRINT " ";RIGHT$("0"+HEX$(D),2); ············· データを出力
210    NEXT J
220    PRINT " :";RIGHT$("0"+HEX$(T),2) ················· チェックサムを出力
230 NEXT I
```

〔図2－20〕

```
1000: CE 00 00 86 0C 17 02 CE 17 03 6B CC 18 00 17 03 :CA  ホ..■...ホ..k7....
1010: 44 17 02 92 30 8D 03 FD 17 03 2D 17 01 ED 1F 01 :18  D..+O■.人..-..O..
1020: 34 10 4F 5F 17 03 2E 17 02 08 30 88 10 4C 81 19 :09  4.O_......Ol.L_.
1030: 26 F1 35 10 10 8E 00 06 17 03 1E 5F 17 02 C9 81 :FA  &円5..■......_../_
1040: 09 10 27 FF C3 81 1B 10 27 01 B6 81 11 10 27 01 :56  ..' テ_..'.カ_..'.
1050: 43 81 08 10 27 00 FB 81 0D 10 27 00 F5 81 2E 10 :77  C_..'.町_..'.略_..
1060: 27 00 B6 81 1C 10 27 00 B0 81 1D 10 27 00 E3 81 :9A  '.カ_..'.ー_..'.ヨ_
1070: 1E 10 27 00 F6 81 1F 10 27 00 B8 34 04 5F 81 2A :1C  ..'.今_..'.74._*
```

085D
086D
087D
088D
089D
08AD
08BD
08CD

```
1080: 27 25 81 2F 27 20 81 2B 27 1B 81 2D 27 16 81 3D :DA  '%_/' _+'._-'._=
1090: 27 11 81 2C 27 0C 81 61 25 11 81 7A 22 0D 88 20 :02  '._,'._a%._z".!
10A0: 20 09 5C 5C 5C 5C 5C CB 41 1F 98 17 02 06 1F 98 :8E   .¥¥¥¥¥ヒA.r.....r
10B0: 35 04 25 88 34 16 54 3A 25 17 E6 E4 58 58 58 58 :24  5.%I4.T:%.■◢XXXX
10C0: E7 E4 A6 84 84 0F AB E0 A7 84 A6 84 17 01 B7 20 :57  ◤◢ヲ■■.ォニァ■ヲ■..キ
10D0: 14 A6 84 84 F0 AB E0 A7 84 A6 84 31 3F 17 02 79 :94  .ヲ■■×ォニァ■ヲ■1?..y
10E0: 17 01 A3 31 21 1F 20 C6 37 17 02 69 5F 4F AE 61 :88  ..」1!. ニ7..i_Oョa
10F0: AB 85 5C C1 10 26 F9 17 01 8C 1F 20 E6 E4 54 3A :B7  ォ■¥チ.&ホ..■. ◥◢T:
1100: CB 3B 17 02 50 A6 84 81 7F 27 04 81 20 24 02 86 :11  ヒ;..Pヲ■_.'._ $.■
1110: 2E 17 01 C2 17 02 42 35 14 5C C1 20 27 0E C5 01 :E4  ...ツ..B5.¥チ '.ナ.
1120: 26 02 31 21 31 21 17 02 30 16 FF 10 1F 20 C6 06 :45  &.1!1!..0. .. ニ.
1130: 1F 02 5F 30 88 10 31 A9 01 00 10 8C 19 00 24 06 :02  .._0I.1ゥ...■..$.
1140: 17 02 16 16 FE F6 17 01 5D 17 00 E6 31 A9 FF 00 :84  ....※今..].. ◥1ゥ .
1150: 20 EE 5A 2B 0E C5 01 27 02 31 3F 31 3F 17 01 F9 :81   /Z+.ナ.'.1?1?..ホ
1160: 16 FE D9 1F 20 C6 34 1F 02 C6 1F 30 10 31 A9 FF :45  .※ル. ニ4..ニ.0.1ゥ
1170: 00 10 8C 00 00 2D 06 17 01 DF 16 FE BF 31 A9 01 :74  ..■..-...°.※ソ1ゥ.
1180: 00 17 02 01 17 02 0E 86 0B 17 01 4A 17 00 A3 17 :05  ........■...J..」.
1190: 01 E4 20 E3 34 76 CC 18 00 17 01 B9 17 01 07 B6 :1C  .◢ ヨ4vフ....ケ...カ
11A0: FD 02 85 02 27 4B 86 04 10 8E 00 00 F6 FD 02 54 :69  人.■.'K■..■..分人.T
11B0: 24 09 31 3F 26 F6 4A 26 EF 20 36 30 8D 02 50 17 :94  $.1?&分J&\ 60■.P.
11C0: 01 86 17 00 46 1F 02 17 00 DC 30 8D 02 52 17 01 :21  .■..F....ワ0■.R..
11D0: 77 30 8D 02 40 17 01 70 17 00 30 34 06 1F 21 CE :8D  w0■.@..p..04..!ホ
11E0: 00 01 AC E4 22 14 17 00 49 17 00 BA 30 88 10 20 :E0  ..ャ◢"...I..コ0I .
11F0: F1 30 8D 02 32 17 01 50 20 02 35 06 35 76 16 FE :66  円0■.2..P .5.5v.※
1200: 07 CC 18 00 17 01 4E 17 00 9C 39 CC 00 00 34 06 :43  .フ....N.. .9フ..4.
1210: 4F 34 06 EC 62 58 49 58 49 58 49 58 49 E3 E1 ED :0C  O4.■bXIXIXIXIヨチロ
1220: E4 17 00 E4 17 00 AF 17 00 8A 24 E4 81 0D 26 F1 :F3  ◢..◢..ッ..■$◢_.&円
1230: 35 86 34 16 1F 10 17 00 4D 1F 98 17 00 48 86 3A :6E  5■4.....M.r..H■:
1240: 17 00 93 34 10 4F 34 02 C6 10 17 00 50 A6 84 17 :F1  ..┣4.O4.ニ...Pヲ■.
1250: 00 34 A6 80 AB E4 A7 E4 5A 26 EF 17 00 3F 86 3A :F9  .4ヲ_ォ◢ァ◢Z&\..?■:
1260: 17 00 73 35 02 17 00 1E 17 00 32 17 00 2F 35 10 :CA  ..s5......2../5.
1270: C6 10 A6 80 81 7F 27 04 81 20 24 02 86 2E 17 00 :B9  ニ.ヲ__.'._ $.■...
1280: 55 5A 26 EE 35 96 34 02 44 44 44 44 17 00 04 35 :24  UZ&/5┃4.DDDD...5
1290: 02 84 0F 81 0A 25 02 8B 07 8B 30 20 39 34 02 86 :A9  .■._.%.■.■0 94.■
12A0: 20 17 00 32 35 82 34 02 86 0D 17 00 29 86 0A 17 :D0   ..25_4.■...)■..
12B0: 00 24 35 82 34 02 81 30 25 18 81 39 22 04 80 30 :8F  .$5_4._0%._9"._0
12C0: 20 0A 81 41 25 0C 81 46 22 08 80 37 1F 89 1C FE :87   ._A%._F"._7.|.※
12D0: 35 82 1A 01 35 82 34 06 11 83 00 00 26 11 17 01 :A6  5_..5_4..■..&...
12E0: 0F B7 FC 84 CC 03 01 FD FC 82 17 01 16 35 86 F6 :70  .キ村■フ..人村_...5■分
12F0: FD 02 C5 02 27 10 54 25 F6 B7 FD 01 C6 00 F7 FD :DB  人.ナ.'.T%分キ人.ニ.秒人
1300: 00 C6 40 F7 FD 00 35 86 34 04 CC 05 00 83 00 01 :42  .ニ@秒人.5■4.フ..■..
1310: 26 FB 17 00 DB CC 29 00 FD FC 82 17 00 E5 17 00 :96  &町..ロフ).人村_..◣..
1320: CF B6 FC 80 8A 80 B7 FC 80 FC FC 83 17 00 D4 5D :01  マカ村_■_キ村_村村■..ヤ]
1330: 27 D8 34 02 C6 81 F7 FD 03 CC 04 00 83 00 01 26 :ED  'リ4.ニ_秒人.フ..■..&
1340: FB C6 00 F7 FD 03 35 86 34 12 A6 80 27 05 17 FF :21  町ニ.秒人.5■4.ヲ_'..
1350: 85 20 F7 35 92 34 06 20 04 34 06 1F 20 17 00 90 :E1  ■ 秒5┤4. .4.. ..┴
1360: B7 FC 86 F7 FC 85 CC 03 03 FD FC 82 86 12 B7 FC :49  キ村■秒村■フ..人村_■.キ村
1370: 84 17 00 8F 35 86 34 06 8D 76 CC 0C 1F FD FC 82 :94  ■..┼5■4.■vフ..人村_
1380: 17 00 80 35 86 34 06 17 00 66 CC 0C 1E FD FC 82 :7A  ..._5■4...fフ..人村_
1390: 17 00 70 35 86 34 56 CC 18 00 17 FF B8 86 20 C6 :EA  ..p5■4Vフ... ク■ ニ
13A0: 4F 17 FF 32 5A 26 FA 8D 47 30 8D 00 10 CE FC 80 :FC  O. 2Z&ヒ■G0■..ホ村_
13B0: C6 33 A6 80 A7 C0 5A 26 F9 8D 48 35 D6 00 00 3F :1E  ニ3ヲ_ァタZ&ホ■H5ヨ..?
13C0: 59 41 4D 41 55 43 48 49 93 D3 8F 90 FC D0 1F B3 :74  YAMAUCHI┣モ┼┴村ミ.ウ
13D0: D0 3B 7D D4 09 FD D0 1F FD D4 0E B7 D4 09 34 10 :08  ミ;}ヤ.人ミ.人ヤ.キヤ.4.
13E0: 8E CF A0 A6 82 A7 88 50 8C C0 00 26 F6 35 10 39 :8A  ■マ ヲ_ァ|P■タ.&分5.9
13F0: 34 02 B6 FD 05 2B FB 86 80 B7 FD 05 B6 FD 05 2A :B5  4.カ人.+町■_キ人.カ人.*
1400: FB 35 82 34 02 7F FD 05 86 18 4A 26 FD 35 82 53 :7E  町5_4..人.■.J&人5_S
1410: 54 41 52 54 20 41 44 44 52 45 53 53 20 3D 24 00 :E2  TART ADDRESS =$.
1420: 45 4E 44 20 20 20 00 50 52 49 4E 54 45 52 20 45 :C0  END   .PRINTER E
1430: 52 52 4F 52 20 21 21 00 00 00 00 00 00 00 00 00 :A7  RROR !!.........
```

第 3 章

マシン語プログラミング part I

——レジスタと基本命令——

1. CPU6809

いよいよ本格的なマシン語学習に入っていくことになります。ここでは、CPUの内部構造を少しみることにします。

図3—1をみてください。CPUには大まかにいうと、命令解読器、演算器、レジスタがあります。**命令解読器**は、メモリから命令を取り出して解読してCPU内の他の部分を順々に制御して命令を実行します（前に説明した制御装置にあたります)。**演算器**は文字どおり演算を行うところですが、ここで注意してほしいことがあります。というのは、演算器は（加算などの場合）2つのデータから1つの結果を出すのですが、この2つのデータは両方同時にメモリから取り出されるわけではなく、また結果も直ちにメモリに格納されるわけではないということです。ここで**レジスタ**が登場します。減算を例に取って説明します。CPUで減算を行うには、まずメモリから引かれる数(被演算数）を取り出してレジスタに格納します。次に演算器によって、レジスタ内の引かれる数から、メモリに格納されている引く数（演算数）を引き、得られた結果をレジスタに格納します。そして必要があればレジスタ内の結果をメモリに格納します。ざっとこのような手順を踏むことになります。すなわちレジスタとは、データを一時的に記憶しておくための場所ということになります（一時的といっても、そのままで消えてなくなるわけではありません。一種のメモリがCPU内にあるというわけです)。

2. レジスタ

それでは6809にはどのようなレジスタがあるのでしょうか。図3—2を見てください。図にあるように、レジスタには多くの種類があります。これらは、それぞれの用途を持っており、個々に長さや使い方があります。ここではそのうちとりあえず知っておかなければならないしレジスタについて説明します。残りのレジスタは追って解説する

図3－1　CPU6809

図3－2　レジスタ

ことにします。

◀アキュームレータA、B▶

この2つのレジスタは8ビットの長さを持っています。主な用途としては演算を行うときのデータや結果の記憶場所としての用途があります。8ビットですから記憶できる数は0から255までで、あまり大きな数の記憶はできません。そこでアキュームレータAとアキュームレータBをつなげて16ビットのレジスタとみなし、0から65535までの数を記憶することもできます。この16ビットのレジスタはアキュームレータDと呼ばれますが、決してアキュームレータA、Bと別のレジスタが存在するわけではありません。単にAレジスタとBレジスタをつないだものにDレジスタという名前をつけただけです。

アキュームレータ(Accumulator)は略してAccと書かれ、アキュームレータAはAccAと略されます。またAレジスタという表現も用いられます。

◀プログラム・カウンタ（PC）▶

このレジスタは他のレジスタとは大きく異なっています(プログラム・カウンタ[Program Counter]ですのでPCと略されます)。

このPCは、次に実行される命令のあるアドレスを保持しています。ですからCPUはPCの内容から命令のあるアドレスを知り、メモリから命令を取り出すわけです。このようにPCはCPUの命令の実行順序を示していく重要なレジスタですので、データの一時記憶場所としての役割は持っていません。

この他のレジスタについては後述することにして先へ進みます。

3. LD命令

いよいよ具体的な命令の解説に入っていくことにします。それではまず当面の課題を提起しましょう。

課題1.

＄6000番地に＄68というデータを格納するプログラムを作成する

まずは課題を分析してみましょう。既に述べたようにCPUには、メモリから2つのデータを取り出して演算しメモリに結果を格納するという一連の動作を、一命令で一気に実行する芸当はできません。常にレジスタを介して演算を行うようになっています。

この課題の場合も同様で、＄68というデータをメモリから取り出して一気にメモリに格納するという命令はありません。ここでもレジスタを介さなければなりません。すなわち、＄68というデータをいったんレジスタに取り込み、次にレジスタからメモリに格納するという2段階の過程を必要とするわけです。BASICのように表記すれば、'＄6000番地＝＄68'ではダメで'レジスタ＝＄68：＄6000番地＝レジスタ'となります。

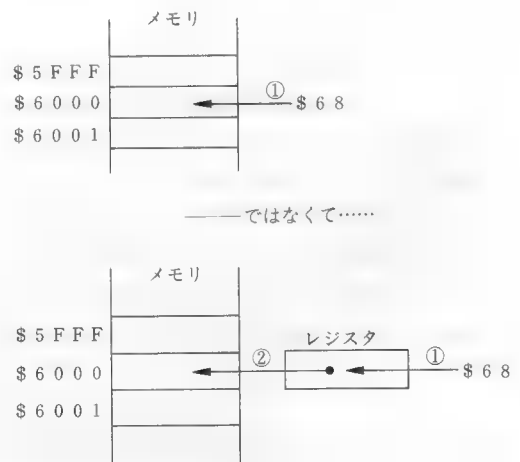



図3―3　課題の分析

それでは2段階の過程の前の部分「＄68というデータをいったんレジスタに取り込む」を考えてみます。そこで登場するのがLD命令です。LD命令というのは、指定した値をレジスタへロードする命令です。ロード(Load)というのは、「カ

セットからプログラムをロードする」というときのロードと同じで、ここでは指定した値を指定したレジスタに取り込む（loadは訳すと「[荷]を積む」です）ことをいいます。

ではLD命令の存在を確認してみましょう。次の図3—4は付録にある**6809インストラクション・コード表**の一部です。このインストラクション・コード表は6809の持っているすべての命令を示したものです。

インストラクション（Instruction：命令）のところにLDがあるのがわかります。次にその右をみてください。「**ニーモニック**（mnemonic）」とあります。これは、「記憶を助ける；記憶術の」ということです。すなわちこれは、命令の動作を記憶するための「記憶術」です。これによって私たちは＄86や＄C6というおもしろみのないマシンコードを覚える必要はなくなるのです。例えば、（まだわからないかもしれませんが）

＄86　＄68

というマシンコードは、Aレジスタに＄68を取り込む命令なのですが、こう書くよりも

LDA　#$68

と書いた方が格段にわかりやすいのです。このようなより人間的な書き方で書かれたものを、**アセンブリ言語**といいます。このアセンブリ言語は、その命令がマシン語の命令(16進のマシンコード)と1対1で対応しているので、アセンブリ言語でプログラムを作るのと、マシンコードでプログラムを作るのとでは、本質的には違いがないわけです。違いがなければわかりやすい方がいいですから、本書でもこれからはアセンブリ言語を使用していきます。

そうすると、マシン語プログラムの開発は次のように行えばよいことになります。

①アセンブリ言語でプログラムを記述する。

②アセンブリ言語を対応するマシンコード（16進数の列）に直す。

③マシンコードを入力して実行する。

実際の例はこれからたくさん出てくるのでそちらに回すことにします。

再び図3—4に戻ると、LD命令には

LDA　LDB　LDD　LDS　LDU
LDX　LDY

の7種類があることがわかります。すなわちLDの後にレジスタ名をつけるとニーモニックになるということです（ですが、LDPC、LDCC、LDDPなどはありません。PCなどは特殊なレジスタだからです）。

ですからニーモニックで

LDA

とは、「Aレジスタに指定した値をロードせよ」ということになります。

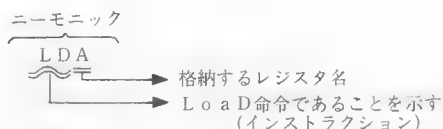



図3—5　LD命令

インストラクション	ニーモニック	アドレッシングモード															動作	5	3	2	1	0
		イミディエイト			ダイレクト			インデックス			エクステンド			インヘレント				H	N	Z	V	C
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#						
LD	LDA	86	2	2	96	4	2	A6	4+	2+	B6	5	3				M→A	●	↕	↕	0	●
	LDB	C6	2	2	D6	4	2	E6	4+	2+	F6	5	3				M→B	●	↕	↕	0	●
	LDD	CC	3	3	DC	5	2	EC	5+	2+	FC	6	3				M:M+1→D	●	↕	↕	0	●
	LDS	10	4	4	10	6	3	10	6+	3+	10	7	4				M:M+1→S	●	↕	↕	0	●
		CE			DE			EE			FE										0	
	LDU	CE	3	3	DE	5	2	EE	5+	2+	FE	6	3				M:M+1→U	●	↕	↕	0	●
	LDX	8E	3	3	9E	5	2	AE	5+	2+	BE	6	3				M:M+1→X	●	↕	↕	0	●
	LDY	10	4	4	10	6	3	10	6+	3+	10	7	4				M:M+1→Y	●	↕	↕	0	●
		8E			9E			AE			BE											

図3—4

4. アドレッシングモード

次に図3―4の中上に「アドレッシングモード」とあるのがわかると思います。

LDA を例にして解説しましょう。既に述べたように、LDA はAレジスタに指定した値をロードする命令ですが、単に「指定した値」といっても、どんな値なのかわかりません。そこでどんな値を指定するかを示す方法が**アドレッシングモード**（Addressing mode）であるわけです。

このアドレッシングモードは6809CPUの特徴ともいえるもので、これを理解すれば、6809CPUの90%は理解できたといっても過言ではないでしょう。これについては、これから順に述べていきますが、取りあえず、どんなアドレッシングモードがあるのか、名前だけあげておきましょう。

①イミディエイトモード*
②ダイレクトモード
③インデックスモード
④エクステンドモード
⑤リラティブモード
⑥インヘレントモード

大まかには上記の6つです。レジスタモードというのもありますが、通常は⑥のイミディエイトモードに含めて考えています。また、③のインデックスモードはさらに細く分けられますが、それは後述ということにします。

*）正確にはイミディエイトアドレッシングモードというように間に「アドレッシング」をはさみます。

5. イミディエイトモード

今までの説明で、課題1の前半部分にはLD命令を用いればよいことがわかったと思います。データの仲介場所としてAレジスタを用いることにすれば、LDA 命令を用いることになります。さて、次にどのような値をロードするのかを指定しなければなりません。

ここでは、$68という定数をAレジスタにロードするので、用いるアドレッシンモードは**イミディエイトモード**（Immediate mode）です。このモードは、後に書いた数値をそのまま、定数として扱うモードです。例を挙げて説明しましょう。課題1の前半は次のようになります。

```
LDA  #$68
```

ここで用いられている“#”は、用いているアドレックシッグモードがイミディエイトモードであることを示す記号です。

同様にXレジスタに$6809という定数をロードするのは、

```
LDX  #$6809
```

となります。

ここで1つだけ注意しておきます。図3―2に示したように、レジスタには各々長さがあり、8ビットのものと16ビットのものがあります。8ビットのレジスタに9ビット以上の情報は入りませんから、

```
LDA  #$432
```

などは不可能です。

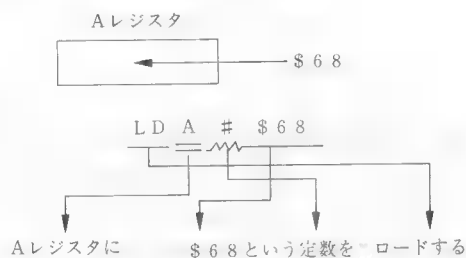



図3―6　イミディエイトモード

6. ST命令

課題1の前半は解決しました。それでは後半の「レジスタからメモリに格納する」を考えましょう。このレジスタからメモリへの格納にはST命令を用います。ST命令は**レジスタの内容をメモリへストアする命令**です。ストア（Store）というのは、「格納する、記憶する」という意味です。ちょうどロードの反対語になります。ST命令にはLD命令と同様に

STA　　STB　　STD　　STS　　STU

STX　　STY

の７種があります。STA はAレジスタの内容をメモリストアへする命令というわけです。

図3―7をみると、ST 命令にはイミディエイトモードがないのがわかります。もしこれがあると一気に定数をメモリに格納できることになり、前述したことと矛盾します。

7. エクステンドモード

ST 命令にはイミディエイトモードはないので他のモードを用いる必要があります。課題１の場合、＄６０００番地にストアするので、用いるモードはエクステンド・モード（Extended mode）です。このモードは後に書いたアドレスに対して直接に処理を施すモードです。例として課題１の後半を挙げると次のようになります。

STA $6000

イミディエイト・モードの時には“#”が付いていましたが、これには付いていません。すなわち記号が何もなければエクステンド・モードであるわけです。実は、

STA >$6000

と書くのが正確なのですが、“>”は省略可能であり、後述するダイレクト・モードと特に区別しなければならないとき以外、省略するのが普通です（本書ではこの“>”は使いません）。

“$”は16進を示すものですからお間違いなく。

エクステンド・モードについて話を進めてきましたがこのモードは別にST 命令に限ったものではありません。既に述べたＬＤ命令にもエクステンドモードがあります。例をあげれば

LDA $6809

は、「Aレジスタに＄６８０９番地の内容をロードする」というわけです。ところで、ここまでレジスタの内容をメモリに格納する命令などみてきましたが、このような命令を実行しても、これらはあくまで、コピーですから、元の場所の値（ST 命令ならレジスタ）は変化しません。

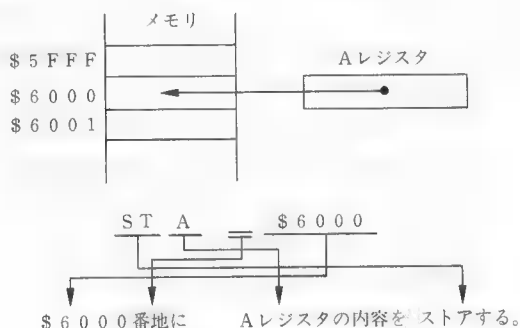



図3―8　ストア命令

8. 16ビットレジスタとメモリ

ところで16ビットのレジスタでエクステンドモードを用いたときに注意しなければならないことがあります。

LDX $6000

インストラクション	ニーモニック	アドレッシングモード															動　作	5	3	2	1	0
		イミディエイト			ダイレクト			インデックス			エクステンド			インヘレント								
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#		H	N	Z	V	C
ST	STA				97	4	2	A7	4+	2+	B7	5	3				A→M	●	↕	↕	0	●
	STB				D7	4	2	E7	4+	2+	F7	5	3				B→M	●	↕	↕	0	●
	STD				DD	5	2	ED	5+	2+	FD	6	3				D→M：M+1	●	↕	↕	0	●
	STS				10	6	3	10	6+	3+	10	7	4				S→M：M+1	●	↕	↕	0	●
					DF			EF			F F											
	STU				DF	5	2	EF	5+	2+	FF	6	3				U→M：M+1	●	↕	↕	0	●
	STX				9F	5	2	AF	5+	2+	BF	6	3				X→M：M+1	●	↕	↕	0	●
	STY				10	6	3	10			10	7	4				Y→M：M+1	●	↕	↕	0	●
					9F			AF	6+	3+	BF											

図3―7

を例に説明しましょう。Xレジスタは16ビットのレジスタですので、当然ロードするデータは16ビットのデータです。しかし、＄６０００番地だけでは８ビットのデータしか得られません。これでは困ります。

そこでＣＰＵは、16ビットのレジスタに関係する場合は、指定したアドレスの内容を上位８ビット、指定したアドレスの次のアドレスの内容を下位８ビットとみなします。図で書けば図3—9のようになります。

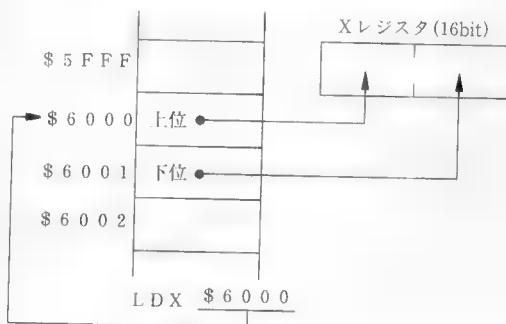



図3—9　16bitレジスタとメモリ

もう１つだけ例をあげます。レジスタの解説で登場したＤレジスタを考えます。このＤレジスタは、Ａレジスタを上位８ビット、Ｂレジスタを下位８ビットとして用いるものです。

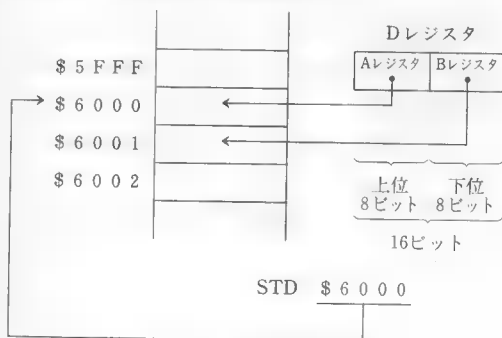



図3－10　Ｄレジスタ

9. ハンドアセンブル

さて、これまでで課題１を達成しました。答えは、

LDA　#$68

STA　$6000

です。これは既に述べたようにアセンブリ言語で記述されています。次にこれを対応するマシンコード（16進数の列）に直す作業をしていきます。この作業を人間が“手”で行うことをハンド・アセンブルといいます。それでは実践してみましょう。

①まず最初の命令'LDA #$68は'は'LDA'という命令でイミディエイト・モードなので、付録のインストラクション表のその項をみます（図3—11）。

インストラクション	ニーモニック	イミディエイト		
		Op	～	#
LD	LDA	８６	2	2
	LDB	Ｃ６	2	2
	LDD	ＣＣ	3	3
	LDS	１０ ＣＥ	4	4
	LDU	ＣＥ	3	3

図3－11　インストラクション表(一部)

②「Op　～　#」とある中のOpの項が、マシン語のＯＰコード（Operation code）です。このＯＰコードがマシン語において命令の種類を指定する部分です。この場合は＄８６が得られます。ＢＡＳＩＣでいうと'ＰＲＩＮＴ　Ａ'の'ＰＲＩＮＴ'に相当します。

③次に'　#$68　'の部分に移ります。この部分はオペランド（operand）と呼ばれて命令の対象を指定する部分です。イミディエイトモードやエクステンドモードの場合にはＯＰコードに続けて、オペランドの数値を置きます。この場合はＯＰコードと合わせて、

＄８６　＄６８

が得られます。

④全く同様にして'　STA　'のエクステンドモードであることからＯＰコードは＄Ｂ７となります。

⑤オペランドも同様ですがアドレスは16ビットなので８ビットずつ分けて、ＯＰコードに続けます。

これで、

＄B 7　＄6 0　＄0 0

が得られます。

他に2つの例を示しました。各自確認してください。注意すべき点は、STY などのようにOPコードが2バイトになる命令もあるということです。

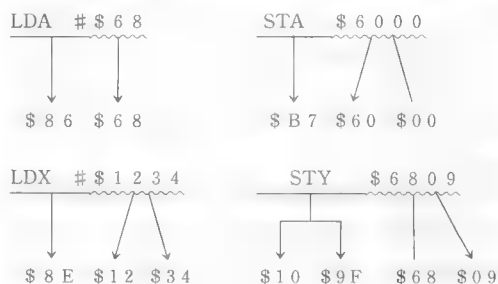



図3－12　ハンドアセンブル

10. メモリマップ

前節までで課題1のマシンコードが作成できましたので実際にFM－7シリーズに入力して動作させることにします。では、どのアドレスに入力したらよいのでしょうか。

まず図3—13をみてください。これはメモリマップと呼ばれるもので、6809の扱うアドレスがFM－7シリーズでどのように使われているかを示したものです。

ＲＯＭ（Read Only Memory）とは読み出し専用のメモリで電源を切ってもその内容が消えないものです。ＲＡＭ（Random Access memory）*2 とは自由に読書きができると同時に電源を切れば内容も消えてしまうというメモリです。

図3—13の内システム領域というのは、FM－7自体が使用しているところで、ユーザ（User：使っている人＝私たち）が使用することはできません。またＲＯＭの部分も書込みができません。さらにＤＩＳＫを接続しているときにはＤＩＳＫ　ＢＡＳＩＣの部分も使用できません。結局残されたのは図中にフリーエリアと書かれた部分だけです。私たちがマシンコードを格納するのはこのフリー

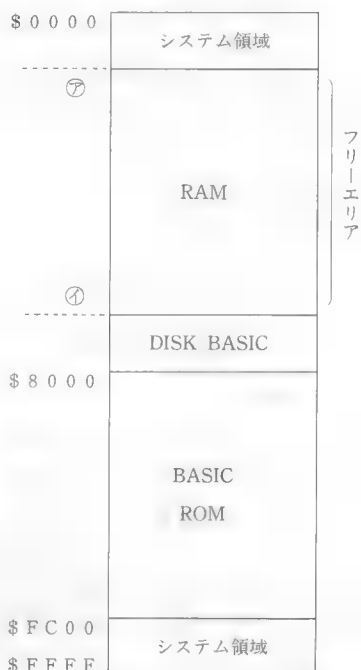



図3－13　メモリ・マップ*1

〔図3－14 フリーエリアの出力〕

```
10 ' ﾌﾘｰｴﾘｱ ｦ ｼｭﾂﾘｮｸ ｽﾙ ﾌﾟﾛｸﾞﾗﾑ
20 A=PEEK(&H33)*256+PEEK(&H34)
30 I=PEEK(&H59D)*256+PEEK(&H59E)-1
40 PRINT "ﾌﾘｰｴﾘｱ ｾﾝﾄｳ ｱﾄﾞﾚｽ ( ｱ )=$";HEX$(A)
50 PRINT "ﾌﾘｰｴﾘｱ ｴﾝﾄﾞ ｱﾄﾞﾚｽ ( ｲ )=$";HEX$(I)
```

図3－13の実行例

```
RUN ⏎
ﾌﾘｰｴﾘｱ ｾﾝﾄｳ ｱﾄﾞﾚｽ ( ｱ )=$BF9 ……………………
ﾌﾘｰｴﾘｱ ｴﾝﾄﾞ ｱﾄﾞﾚｽ ( ｲ )=$71D3 …………………… } 筆者のシステムの場合

Ready
```

エリアの部分でなければいけないわけです。

この図には㋐と㋑のアドレスが書いてありませんが、これは、DISKの接続状況などによって違いがあるためです。これらのアドレスは図3—14のプログラムを走らせることによって得られるので、一度走らせて図中に記入しておくとよいでしょう。

これでアドレスを決定できます。ここでは＄5000番地から入力することにします。

アドレス	メモリ	
＄5000	＄86	LDA　#＄68
＄5001	＄68	
＄5002	＄B7	STA　＄6000
＄5003	＄60	
＄5004	＄00	
＄5005		

図3－15　課題1のプログラム

＊1）このメモリマップは概略であって詳細ではありません。詳細はメモリマップは、文献3の付録などを参照してください。また、実践編でももう一度解説します。
＊2）本来RAMとは、「随時、参照ができるメモリ(本文中の意味でのRAMやROMを含む)」のことをさしていましたが、最近では「読書きができるメモリ」のことをさすようになってきました。本書では後者の意味で用いています。

11. CPUの動作

課題1はいよいよ実行を残すだけになりましたが、実行の前に紙上でCPUの動作をたどってみましょう（まだ実行しないでください)。

まずモニタで'G5000⏎'とするとPC(プログラムカウンタ）に＄5000が代入されてプログラムの実行が開始されいます。このPCは次に実行すべき命令のアドレスを保持しているのでCPUはPCの内容（＝＄5000）をアドレスバスへ出力し＄5000番地を指定して命令をメモリから取り出します（図3—16)。そしてその命令を解読して、

①イミディエイトモードであること。

②Aレジスタにロードする命令であること。

であることを知り、データ（定数）をロードします。このときPCは命令を読み取るごとに＋1されており、LDA #$68 を実行し終るとPCは＄5002になっています。次に同様にしてAレジスタの内容を＄6000番にストアします。

STA$6000 を実行し終ると次にCPUはどうなるのでしょうか。私たちはここでCPUが実行を終了してくれることを望んでいるのですがCPUにはそれがわかりません。ですからCPUは＄5005番地から実行を続行します。しかし私たちには＄5005番地以降はどんなメモリ内容であるかわかりません。そうなるとCPUは私たちの意向を無視して、動作することになります。このようになることを**暴走**と呼びます。

それでは暴走させないためにはどうしたらよいでしょうか。ここで、

JMP $ABF4

という命令を使用します。新しい命令が出てきましたが、これについては後述することにして、今は一種のおまじないとして考えていてください。この命令は実行すると、プログラムの実行を終了しモニタに制御を移すものです。これから当分の間、プログラムの終りにはこの命令をつけ加えるようにします。

アドレス	メモリ	
＄5000	＄86	LDA　#＄68
＄5001	＄68	
＄5002	＄B7	STA　＄6000
＄5003	＄60	
＄5004	＄00	
＄5005	＄7E	JMP　＄ABF4
＄5006	＄AB	
＄5007	＄F4	

図3－17　課題1の完成プログラム

メモリの動作
アドレス
メモリの内容
CPUの動作
動作
PC
Aレジスタ
アドレスバス
$5000
$5000
?
$5000
$86
データバス
$86
命令解読
イミディエイトモード
Aレジスタにロードする
アドレスバス
$5001
$5001
$5001
$68
データバス
$68
Aレジスタへロード
$68
命令終了。次へ
アドレスバス
$5002
$5002
$68
$5002
$B7
データバス
$B7
命令解読
エクステンドモード
Aレンジスタをストアする
アドレスバス
$5003
$5003
$5003
$60
データバス
$60
一時記憶
アドレスバス
$5004
$5004
$5004
$00
データバス
$00
一時記憶
アドレスバス
$6000
$6000
$68
アドレスバス
$68
Aレジスタをストア
$68
命令終了。次へ
アドレスバス
$5005
$5005
$5005
$XX
命令解読
?
メチャクチャな動作となる

図3—16 CPUの動作

よってこの命令をハンドアセンブルすると
＄7E　＄AB　＄F4
となります（JMP命令のエクステンドモード）。

結局、課題1のプログラムは
LDA　#$68
STA　$6000
JMP　$ABF4
となり、これを$5000番地からハンドアセンブルすると図3—17のようになります。

〔図3－18　入力〕

```
MON

*M5000
5000 00-86 ⏎ ……メモリへ書き込む
5001 00-68 ⏎
5002 00-B7 ⏎
5003 00-60 ⏎
5004 00-00 ⏎
5005 00-7E ⏎
5006 00-AB ⏎
5007 00-F4 ⏎
5008 00-. ⏎

*■
```

〔図3－19　確認〕

```
*D5000 ⏎ ……確認

5000 86 68 B7 60 00 7E AB F4
5008 00 00 00 00 00 00 00 00
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

〔図3－20　実行〕

```
*M6000 ⏎ ……実行前
6000 00-. ⏎ ……＄6000番地は0

*G5000 ⏎ ……プログラムの実行

*M6000 ⏎
6000 68-■ ……＄6000番地は＄68になっている

*■
```

それではさっそく実行してみることにします。入力から実行までの過程を図3—18～図3—20に示します。

実行の結果確かに＄6000番地に＄68が格納されているのがわかります。

もし実行した時に異常な動作が生じた場合は、BREAKキーを押しながら、本体後部のリセットスイッチを押してみてください。これで

```
Abort
Ready
```

と出力されればBASICに戻れたことを示しています。入力したプログラムは残っていると思いますので間違いを捜してください。

Abortのメッセージが出力されない場合には、リセットスイッチを単独で押してください。しかしこの場合には残念ながら入力したプログラムは失われます。

12. Rサブコマンド

ここで、モニタの使用法のところでやり残していた、Rサブコマンドについて解説します。

このRコマンドは、モニタに制御が移ったときの各レジスタの値を表示したり、Gコマンドで実行を開始する際にレジスタに値を設定したりするサブコマンドです。実際の例で使用法を述べます。課題1のプログラムは、Aレジスタを使用するプログラムでしたので、実行前と実行後ではAレジスタの値は違っているはずです。実行後にはAレ

〔図3－21　Rコマンド①〕

```
*R ⏎ ……実行前にRコマンドでレジスタの値を確認
CC 04- ⏎
A  00- ⏎  ……Aレジスタは0
B  44- ⏎
DP 00- ⏎
X  ABF4- ⏎
Y  C4E5- ⏎  ……使用法はMコマンドと同じ
U  0340- ⏎
PC ABF9- ⏎

*■
```

ジスタには＄６８が入っているはずです。これを確認しましょう。

まず実行前の各レジスタの内容をみてみましょう。

Aレジスタは0になっていることがわかります。

次に実行後の各レジスタを見てみると、たしかにAレジスタには＄６８が入っていることがわかります。

〔図3－22　Rコマンド②〕

```
*G5000 ⏎ ........................ プログラム（課題１の実行）

*R ⏎
CC  00-
A   68-  ............ Aレジスタが＄６８にかわっている
■   44-
DP  00-
X   ABF4-
Y   C4E5-
U   0340-
PC  ABF9- ⏎
*■
```

Aレジスタの他にCCレジスタも変化していますがこれについては後述することにします。

これでモニタのすべてのサブコマンドが使えるようになりました。

第 4 章

マシン語プログラミング part II

——加算と減算——

1. アセンブラ

私たちはこれまで、プログラムを開発するのに、まずアセンブリ言語で記述し、それをハンドアセンブルという方法を用いて“手”でマシンコードへと直していました。この方法は、マシン語の学習には非常によい方法といえるのですが、短いプログラムならまだしも長いプログラムになると、大きな労力を必要とすると同時に、誤りの入り込む余地も大きくなってきます。

このアセンブルの作業は規則的であり、かつ正確さを要求される作業ですから、コンピュータ自身にその作業を行わせるのが最適です。この作業を実現するプログラムが**アセンブラ**と呼ばれるプログラムです。このアセンブラを用いれば、人間は、アセンブリ言語を入力するだけでマシンコードを得ることができるわけです。

アセンブラは各ＣＰＵごとに発売されており、またＦＭ－７シリーズ上で走る6809用アセンブラも、数多く発売されています。そのうち主なアセンブラの使用法は、実践編で述べることにして、ここでは、本書で使用したアセンブラ*による出力の見方だけを述べておきます。

図4－1をみてください。これがアセンブラによって出力された、**アセンブルリスト**です。このうち人間が入力したのは④、⑤の部分で、①～③が得られたマシンコードです。もう少し詳しく解説しましょう。

①は、マシンコードのアドレスを示しています。例えば＄Ｂ７は＄５００２番地に格納されることを示しています。

②は、アセンブリ言語のニーモニック④から得られたマシンコードのうち、ＯＰコードの部分を示しています。

③は、アセンブリ言語のオペランド部（⑤）を変換した結果を示しています。

④は、アセンブリ言語のニーモニックの部分を、⑤は、オペランドの部分を示しています。

次の⑥と⑦は、人間がアセンブラに対して指示を与える命令で、マシンコードには変換されませ

ん。このうち⑥は、マシンコードを＄５０００番地から格納することを示しており、⑦はアセンブラにこれで終りということを指示する命令です。これらについては実践編で詳しく解説します。

⑧は、アセンブルの結果、文法間違いがいくつあったかを示したものです。

本書ではこれからプログラムをこのアセンブルリストの形で示していくことにします。アセンブルリストのうち、実行させる際に入力しなければならないマシンコードは、上述したように図中の②と③の部分に出力されています。ですから図中の網のかかった部分をその左のアドレスにそって入力していけばよいことになります（当分はアセンブルリストと同時にダンプリストも示しますので、それをみて確認するとよいでしょう）。

＊ＦＬＥＸというＤＯＳ上のアセンブラを使用しています。

2. 加算

さて、この章ではいよいよコンピュータらしい動作、「計算」を取りあげます。6809ＣＰＵは、算術演算をする命令としては、加算命令、減算命令、乗算命令があります。なぜ除算命令がないのかというと除算命令を実現するには複雑な内部回路が必要だからです。しかし除算は減算の繰返しによって実現できるので問題ありません。

それだけではありません。実は後述するように減算命令は加算命令によって代行でき、乗算命令も加算命令の繰返しによって実現できるので、すべての算術演算は加算を基本に成立しています。極端にいえば加算命令だけあれば十分ともいえます。ですから、まずその加算命令を追ってみることにしましょう。

課題２

＄６８＋＄０９

を計算して＄６０００番地に結果を格納するプログラムを作成する。

まずこの課題をやる前に、実際に手で計算してみましょう。これにはいくつかの方法がありますのでひとつひとつ述べていきましょう。

＄６８	⟶	６×16＋８＝ １０４ｄ
＋＄０９	⟶	０×16＋９＝ ９ｄ
＄７１	⟵	７×16＋１＝ １１３ｄ

図４－２　10進数になおして計算

図４－３　直接16進数で計算

＄６８	⟶	％０１１０ １０００
＋＄０９	⟶	％００００ １００１
＄７１	⟵	％０１１１ ０００１

（繰りあがり）

図４－４　２進数で計算

第1番目の方法は16進数を10進数に変換して計算して、最後に16進数に戻す方法です。この方法を図4—2に示します。

2番目の方法は直接16進数で計算する方法です。16進数は16で1繰りあがるので図4—3のようになります。

3番目の方法は、16進数をコンピュータの基本である2進数に直して計算する方法です。2進数は2で1繰りあがる（1の次は10）ので図4—4になります。

いずれの方法でも答は＄７１（＝113ｄ）であることがわかります。

3. ADD命令

それでは実際のプログラミングに移りましょう。加算を実現する命令はADD命令です。ADD命令は、**指定レジスタに指定した値を加える命令**です。指定できるレジスタはアキュームレータであるA、B、Dの各レジスタです。A、Bレジスタを用いた場合は8ビット、Dレジスタを用いた場合は16ビットの加算ができます。

値の指定には、既に述べたようにアドレッシングモードを用います。ADD命令にはイミディエイトモードも、エクステンドモードもあるのでレジスタに定数を加えることも、どこかのアドレスの内容を加えることもできることがわかります。

課題2を分析してみましょう。まず演算を行うにはあらかじめ被演算数（＋や－などの演算子の左側にくる数、この場合＄６８）をレジスタに格納しておかなければなりません。次に演算を行いますが、課題の場合、演算数（演算子の右側にくる数。この場合＄０９）は定数ですのでイミディエイトモードを用いればよいことになります。演算の結果は再びレジスタに格納されますので、最後にそれを＄６０００番地にストアします。

これでプログラムが組めるようになりました。

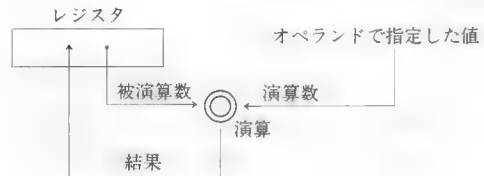



図４－５　演算のしくみ

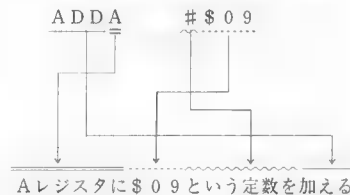



図４－６　ＡＤＤ命令

〔図４－８　課題２のダンプリスト〕

```
*D5000

5000 86 68 8B 09 B7 60 00 7E
5008 AB F4 00 00 00 00 00 00
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

〔図４－７　課題２〕

```
5000                ORG   $5000 ……………＄５０００番地からプログラムを格納する
5000 86    68       LDA   #$68  ……………Ａレジスタに＄６８という定数をロードする
5002 8B    09       ADDA  #$09  ……………Ａレジスタに＄０９という定数を加える
5004 B7    6000     STA   $6000 ………＄６０００番地にＡレジスタの内容をストアする
5007 7E    ABF4     JMP   $ABF4 ……………………………………………………モニタへ
                    END   ………………………………………………………………終り

0 ERROR(S) DETECTED
```

アセンブルリストを図4―7に示します。あなたの考えたプログラムと一致しているでしょうか。図4―8にダンプリスト、図4―9に実行の様子を示します。確かに手で計算したのと同じ値＄７１が、＄６０００番地にストアされています。

4. *16ビットの加算*

では、もう少し進んで16ビットの加算を考えてみます。課題２では演算をＡレジスタを用いて行いましたが、Ｄレジスタを用いて演算すれば、16ビットの加算ができます。

図4―10にこれを実現したものを示します。このプログラムは、＄６０００、＄６００１番地の内容と＄６００２、＄６００３番地の内容を加えて、結果を＄６００４、＄６００５番地に格納するプログラムです。

このプログラムを用いて

＄０１２３＋＄４５６７

を計算した実行例を図4―13に示します。結果を確認してください。

ここで１つ便利なツール（道具）を紹介します。図4―14をみてください。これは図4―10のプログラムを１命令ずつ実行していったとき、レジスタの値がどう変っていくかを見たものです。これを出力したツールは「ＦＬＥＸ Debug Pack」と呼ばれるプログラムです。このツールに関する詳しい説明は省きますが、この出力をみれば、どのような実行のされかたをしているのかがひと目でわかります。図4―15は図4―7（課題２）の実行の様子です。

このツールを使うと理解しやすいので今後も折に触れて使用します。

〔図４－９　課題２の実行〕

```
*M6000
6000 00-.  ……$６０００番地は０

*G5000  ……実行

*M6000  ……実行終了
6000 71-■ …$６０００番地は$７１に変わっている
```

〔図４－10　16ビットの加算〕

```
 5000                  ORG    $5000
 5000 FC    6000       LDD    $6000   ……①：$６０００，$６００１番地をDレジスタにロード
 5003 F3    6002       ADDD   $6002   ……②：Dレジスタに$６００２，$６００３番地の内容を加える
 5006 FD    6004       STD    $6004   ……③：$６００４，$６００５番地にDレジスタをストアする
 5009 7E    ABF4       JMP    $ABF4
                       END

0 ERROR(S) DETECTED
```

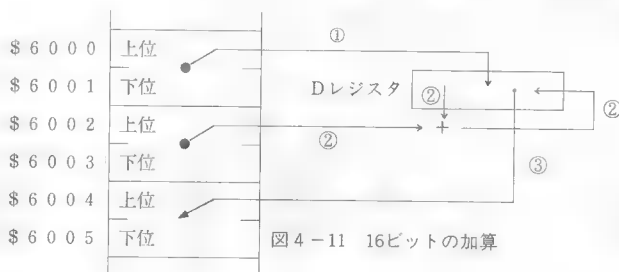



図４－11　16ビットの加算

図４－11　16ビットの加算の図

〔図4－12　16ビットの加算・ダンプリスト〕

```
*D5000

5000 FC 60 00 F3 60 02 FD 60
5008 04 7E AB F4 00 00 00 00
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

〔図4－13　16ビットの加算・実行例〕

```
*M6000 ……………………………………値をセット
6000 00-01 }
6001 00-23 } ………………………………$0123
6002 00-45 }
6003 00-67 } ………………………………+$4567
6004 00-   }
6005 00-.  } ………………………………結果の入るところ
*G5000 ……………………………………………実行
*M6000 ……………………………………結果の確認
6000 01- }
6001 23- } ………………………………$0123
6002 45- }
6003 67- } ………………………………+$4567
6004 46- }
6005 8A- } ………………………………$468A
*■
```

〔図4－14　実行の様子（図4－10のプログラム）〕

```
右の命令を実行する前の各レジスタの内容                          ここでは  PCの内容
                                                                関係ない  （実行前）
                                                                   ↓        ↓実行しようとする命令
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDD  $6000
CC=00 A=01 B=23 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 ADDD $6002
CC=00 A=46 B=8A DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5006 STD  $6004
CC=00 A=46 B=8A DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5009 JMP  $ABF4
REF-MP TRAP AT 5009 ←モニタへ移ったことを示す
**
      これで見るとDレジスタがAレジスタとBレジスタで成り立っていることがよくわかる
```

〔図4－15　実行の様子（図4－7のプログラム）〕

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  #$68
                         Aレジスタにロードする
CC=00 A=68 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5002 ADDA #$09
この変化                 Aレジスタに$09を加える
は後述
CC=20 A=71 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5004 STA  $6000
                         Aレジスタを$6000番地にストア
CC=20 A=71 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 JMP  $ABF4
                         モニタへ移る
REF-MP TRAP AT 5007 ←$5007番地からの命令でモニタへ移ったことを示す
```

5. Cフラグ

図4―16のプログラムを見てください。このプログラムは課題２の変形で

＄ＡＢ＋＄ＣＤ

を計算し、その結果を＄６０００番地に格納するプログラムであることはわかると思います。まずは実行する前に手で計算してみます（図4―17）。

計算の結果は残念ながら８ビットでは収りません。このような場合ＣＰＵはどう処理するのでしょうか。実行してみると、図4―18のようになります。

この実行例で説明しましょう。

ＣＰＵは得られた９ビットの答（決して10ビット以上にはなりません。＄ＦＦ＋＄ＦＦ＝＄１ＦＥが最大です）のうち、下位８ビットを指定したレジスタ（この例ではＡレジスタ）に格納して、繰りあがりで生じた９ビット目をＣＣレジスタの一番右のビット（最下位ビット＝ビット０）に格納します。このＣＣレジスタのビット０は、Ｃフラグ（キャリーフラグ）と呼ばれて、計算の結果、

```
  1     1  111    ←――繰りあがり
   % 1010 1011
 + % 1100 1101
 ---------------
  % 10111 1000
        ||
     $178
```

図４－17　手で計算

```
      %00010010………$12
   +  %00110100………$34
 ------------------------
 %  0  01000110………$46
    ↑  └──────┘
 Cフラグ   レジスタ
```

①繰り上がりなし

```
      %10001001………$89
      %10101011………$AB
 ------------------------
 %  1  00110100……$134
    ↑  └──────┘
 Cフラグ   レジスタ
```

②繰り上がりあり

図４－19　Ｃフラグの２つのケース

〔図４－16　＄ＡＢ＋＄ＣＤ〕

```
5000                    ORG   $5000
5000 86   AB            LDA   #$AB
5002 8B   CD            ADDA  #$CD
5004 B7   6000          STA   $6000
5007 7E   ABF4          JMP   $ABF4

0 ERROR(S) DETECTED
```

〔図４－18　＄ＡＢ＋＄ＣＤの実行〕

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  #$AB
CC=08 A=AB B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5002 ADDA #$CD
      ||
%00001000
      Cフラグ↓0から1になった  +
%00101011
      ||
CC=23 A=78 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5004 STA  $6000
CC=21 A=78 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 JMP  $ABF4
REF-MP TRAP AT 5007
**
```

ビット7から繰りあがりが生じたときにはセットされ(1になる)、繰りあがりが生じなかったときにはリセット（0になる）されるようになっています。

これはみかたを変えると、指定したレジスタのビット8（9ビットめ）が存在して、それがCCレジスタ内のCフラグとも考えることができます。

2つのケースを図4—19にあげておきます。

6. ADC命令

次にCフラグの利用法を探ってみます。

私たちは小学校で桁数の多い加算を習いました。その過程は、まず1桁＋1桁を学習し、次に「繰りあがり」を学習するという順序でした。

```
Aレジスタ  Bレジスタ
     0 ←┐
  $01   │ 23
 +$45   │ 67
 ───────┼─────
  $46   └─8A
     繰りあげなし
```

①繰りあげのない場合

```
Aレジスタ   Bレジスタ
     1 ←┐
  $65   │  86
 +$68   │  88
 ───────┼──────
  $CE   └─0E
     繰りあげあり
```

② 繰り上げのある場合

図4—20　繰りあがりを利用する

既に私たちは6809のマシン語で、加算の基本命令として ADD命令、繰りあがりに対応するCフラグを知っています。これに今から説明する ADC命令を加えると、桁数に制限されない加算ができるようになります。例として、図4—11のプログラムをDレジスタとしてではなく、Aレジスタと

〔図4—23　16ビットの加算（モニタによる実行)〕

```
*D5000

5000 B6 60 00 F6 60 01 FB 60
5008 03 B9 60 02 B7 60 04 F7
5010 60 05 7E AB F4 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

〔ダンプリスト〕

```
*M6000
6000 00-65 }
6001 00-86 } ………被演算数をセット
6002 00-68 }
6003 00-88 } ………演算数をセット
6004 00-   }
6005 00-.  } ………初めは0

*■


*G5000 ……………………………実行

*M6000
6000 65-
6001 86-
6002 68-
6003 88-
6004 CE-   }
6005 0E-■ } ………結果　$CE0E
```

〔図4—21　A・Bレジスタを用いた16ビット加算〕

```
5000                 ORG   $5000
5000 B6   6000       LDA   $6000 ………上位8ビットをAレジスタにロード
5003 F6   6001       LDB   $6001 ………下位8ビットをBレジスタにロード
5006 FB   6003       ADDB  $6003 ………下位8ビットの加算
5009 B9   6002       ADCA  $6002 ………上位8ビットをCフラグとともに加算
500C B7   6004       STA   $6004 ………上位8ビットを格納
500F F7   6005       STB   $6005 ………下位8ビットを格納
5012 7E   ABF4       JMP   $ABF4
                     END

0 ERROR(S) DETECTED

SYMBOL TABLE:
```

Bレジスタとして計算してみることにします。

手順は人間がやる方法と同じです。まず下位8ビットをBレジスタを用いて加算します。するとCフラグが繰りあがりの有無に応じて、セットまたはリセットされます。次に上位8ビットをAレジスタを用いて加算するわけですが、ここでは下位8ビットからの繰り上がり＝Cフラグを含めて加算しなければなりません。このCフラグ（＝下位からの繰りあがり）を含めて加算する命令がADC命令です（図4—20)。

この手順をプログラムにしたものが図4—21です。そして、これを図4—20のデータで実験した物が図4—22です。ADC命令が確かに、Cフラグを含めて加算しているのがわかると思います。

ADC命令はCフラグを含めて加算すること以外はADD命令と同じですから、ADC命令実行でさらに繰りあがりが生じればCフラグはセットされ、生じなければリセットされます。これを応用することにより何桁の加算でも実現できるのがわかります。

7. SUB命令

加算の次は当然減算ということになります。

図4—24のプログラムをみてください。このプログラムは、＄６０００番地の内容から＄６００１番地の内容を引いて、その結果を＄６００２番地に格納するプログラムです。復習を兼ねてこのプログラムを１命令ずつ解説しましょう。

まず

LDA $6000

〔図４－22　16ビットの加算・実行例〕

```
**
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  $6000
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 LDB  $6001
CC=00 A=01 B=23 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5006 ADDB $6003
%00000000                   $6003番地
        ↓くり上がり  +←  $67
%00001010  なし
CC=0A A=01 B=8A DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5009 ADCA $6002
 Cフラグ  ↓
    0 ──→ + ←── $6002番地
          ↓       $45
CC=00 A=46 B=8A DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500C STA  $6004
CC=00 A=46 B=8A DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500F STB  $6005
CC=08 A=46 B=8A DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5012 JMP  $ABF4
REF-MP TRAP AT 5012
**                          ① $0123+$4567


**
**
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  $6000
CC=00 A=65 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 LDB  $6001
CC=08 A=65 B=86 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5006 ADDB $6003
%00001000                       $6003番地
        ↓くり上がり      +←──  $67
%00000011  あり
CC=03 A=65 B=0E DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5009 ADCA $6002
Cフラグ  ↓        $6002番地
  1  ──→ + ←──   $68
         ↓
CC=0A A=CE B=0E DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500C STA  $6004
CC=08 A=CE B=0E DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500F STB  $6005
CC=00 A=CE B=0E DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5012 JMP  $ABF4
REF-MP TRAP AT 5012
**                          ② $6586+$6888
**
**
```

で、＄６０００の内容をAレジスタにロードします。この値が被演算数（引かれる数）となります。

次は

SUBA $6001

です。ここに出てきたSUB命令は、指定したレジスタから指定した値を引く命令です。ですからここでは図4—25に示すような命令となります。このSUB命令でもADDやADC命令と同様に演算の結果は、もとの指定したレジスタに再格納されます。

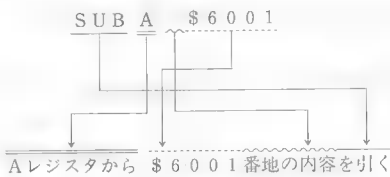



図4－25　SUB命令

〔図4－26　SUB命令実行例(1)〕

```
*D5000

5000 B6 60 00 B0 60 01 B7 60
5008 02 7E AB F4 00 00 00 00
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
          〔ダンプリスト〕

*M6000
6000 00-68 ……………………引かれる数
6001 00-09 ……………………引く数
6002 00-. ……………………結果の入るところ

*G5000…………実行

*M6000
6000 68-
6001 09-
6002 5F-■……………………$68－$09＝$5F
```

次の

STA $6002

は、＄６００２番地にAレジスタの内容をストアする(格納する)命令です。これにより結果が＄６００２番地に格納されることになります。

JMP　$ABF4

これは、既に述べたように「暴走させないためのおまじない」です。

まずはこのプログラムで

＄６８－＄０９

を計算させましょう。実行例を図4—26に示します。

8. SBC命令

８ビットの減算の次は16ビットの減算を行ってみます。本章７節では、Aレジスタを用いて８ビットの減算を行いましたが、Dレジスタを用いることにより16ビットの減算ができます。

図4—27のプログラムをみてください。このプログラムは＄６０００、＄６００１番地の内容から＄６００２、＄６００３番地の内容を引き、結果を＄６００４、＄６００５番地に格納するプログラムです。図4—28はその実行例です。

次に加算のときと同様に、Dレジスタではなく、AレジスタとBレジスタを用いて８ビットずつ計

〔図4－24　減算命令〕

```
5000                  ORG   $5000
5000 B6    6000       LDA   $6000
5003 B0    6001       SUBA  $6001
5006 B7    6002       STA   $6002
5009 7E    ABF4       JMP   $ABF4
                      END

0 ERROR(S) DETECTED

SYMBOL TABLE:
```

算するプログラムを作成しましょう。

例として

〔図4－28　16ビットの減算・実行例〕

```
*D5000

5000 FC 60 00 B3 60 02 FD 60
5008 04 7E AB F4 00 00 00 00
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

〔ダンプリスト〕

```
*M6000
6000 00-43 }
6001 00-21 } ………………数をセット
6002 00-12 }               $4321－$1234
6003 00-34 }
6004 00-
6005 00-.

*G5000…………………………………………実行

*M6000
6000 43-
6001 21-
6002 12-
6003 34-
6004 30- } ……………………答　$30ED
6005 ED-■ }

Break
```

$4321－$1234

で考えてみましょう。

まずは下位8ビットをBレジスタを用いて計算します。つまり

$21－$34

を計算するわけです。しかしこのままでは引けないので、**上位8ビットから1だけ借りてきて、**

$121－$34＝$ED

となります。実は、この「上位8ビットから1だけ借りてくる」という動作は、CPUの側で自動

```
借り ──→ 10 ───────── $100
  │       00100001 ─ $21
  │   －)  00110100 ─ $34
  │       11101101 ─ $ED
  └─→ [1]
```

①$21－$34（借りの生じるケース）

```
      01101000 ─ $68
 －)  00001001 ─ $09
      01011111 ─ $5F
   [0] ←─借りなし
```

②$68－$09（借りの生じないケース）

図4－29　借りる場合と借りない場合

〔図4－27　16ビットの減算〕

```
5000                ORG    $5000
5000 FC   6000      LDD    $6000
5003 B3   6002      SUBD   $6002
5006 FD   6004      STD    $6004
5009 7E   ABF4      JMP    $ABF4
                    END

0 ERROR(S) DETECTED

SYMBOL TABLE:
```

図4－30　$4312－$1234

的に行われています。借り(ボロー)が生じたときにはCフラグがセットされ、借りが生じなかったときにはCフラグはリセットされます（図4—29)。

次に上位8ビットの計算に移ります。ここでは借りを考慮しなければならないので、SUB命令は使えません。そこでSBC命令を用います。このSBC命令は、SUB命令と同じ動作の後、さらに下位への貸しを引く命令です。すなわちCフラグが0（＝借りなし）のときにはＳＵＢ命令と同じで、Cフラグが1（＝借りあり）のときにはさらに1を引くことになります（図4—30)。

以上をプログラムにしたものが図4—31です。図4—28の実行結果とこのプログラムの実行結果(図4—32)を比べると同じ答が出ているのがわかります。

〔図4－32 A・Bレジスタを用いた16ビット減算実行例〕

```
*D5000

5000 B6 60 00 F6 60 01 F0 60
5008 03 B2 60 02 B7 60 04 F7
5010 60 05 7E AB F4 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
            〔ダンプリスト〕

*M6000
6000 00-43 }
6001 00-21 }                数をセット
6002 00-12 } ...............＄４３２１－＄１２３４
6003 00-34 }
6004 00-
6005 00-.

*G5000...................................実行

*M6000
6000 43-
6001 21-
6002 12-
6003 34-
6004 30- } ......................答　＄３０ＥＤ
6005 ED-■ }
```

9. 補数

前節の例では正しく減算ができました。では次に、減算の結果が負になるように数を与えた場合にはどうなるでしょうか。

例として、

＄０９－＄６８

を考えます。＄０９＝9、＄６８＝104ですから答えは－95になるはずです。ＣＰＵに実行させた様子を図4—33に示します。答えは＄Ａ１であることを示しています。

＄Ａ１＝161≠－95

ですから、ＣＰＵは答が負になる減算はできないと思ってしまいますが、実は、別の解釈をすれば、＄Ａ１＝－95となり答は正確に求められているのです。

今まで私たちは正の数しか扱っていませんでしたが、図4—34のように数を対応させると、1バイトで－128～＋127までの数を表すことができます。この図4—34のように数を表現することを**2の補数表現**といいます。また、今までどおり1バイトで

〔図4－31　Ａ・Ｂレジスタを用いた16ビットの減算〕

```
5000                 ORG   $5000
5000 B6    6000      LDA   $6000..............上位8ビットをAレジスタにロード
5003 F6    6001      LDB   $6001..............下位8ビットをBレジスタにロード
5006 F0    6003      SUBB  $6003..............下位8ビットの減算
5009 B2    6002      SBCA  $6002..............上位8ビットの減算（Cフラグを含む）
500C B7    6004      STA   $6004..............上位8ビットを格納
500F F7    6005      STB   $6005..............下位8ビットを格納
5012 7E    ABF4      JMP   $ABF4..............オマジナイ
                     END

0 ERROR(S) DETECTED
```

0～255までの数を表す表現のことを**絶対値表現**といいます。

では次に2の補数表現というのは、どういう表現法なのかを解説しましょう。2の補数表現（正確には2^8の補数表現といいますが、これからは単に補数表現と略します）というのは、1バイト中の一番上の位（ビット7）を符号を示すビットとして、正負の数を表現する方法です。＄Ａ１を例に取りましょう。＄Ａ１は2進数で書けば、

＄Ａ１＝％１０１０００１

となりビット7は1なので負の数を示していることになります。次にこの＄Ａ１がどんな負の数を示しているのかをみましょう。これをみる方法には3種類ありますが、どれか1つの変換法を知っていれば十分です。＄Ａ１を例に取ります。

〔図４－33　＄０９－＄６８の計算〕

```
プログラムは図4－24と同じ
*M6000
6000 00-09 }
6001 00-68 }  ……………$09－$68をセットする
6002 00-00
6003 00-.

*G5000……………………………………………
                                        実行
*M6000
6000 09-
6001 68-       $09－$68＝$A1?
6002 A1-■
```

10進	16進	2進
－128	＄80	1000　0000
－127	＄81	1000　0001
	⋮	
－3	＄FD	1111　1101
－2	＄FE	1111　1110
－1	＄FF	1111　1111
0	＄00	0000　0000
1	＄01	0000　0001
	⋮	
126	＄7E	0111　1110
127	＄7F	0111　1111

図４－34　８ビットによる２の補数表示

☆方法１☆

％１０００００００００＝2^8＝＄１００から、＄Ａ１＝％１０１０００１を引きます（図4―36①)。すると、％０１０１１１１１＝＄５Ｆ＝95が得られます。

この方法は最も補数の意味にそったやり方です。というのは、補数というのはある数αについて、

$\alpha+\beta=2^8$（16ビットのときは2^{16}）

```
            ┌── 1なので負の数
            ↓
$A1 ──→ % 1010  0001
            ↑
            │
        最上位ビット(ビット7)
```

図４－35　＄Ａ１を２進数で

```
    %100000000…$100
－) %  10100001……$A1……（－95）
    %  01011111……$5F＝95
                        ↓
  ①2⁸から引く方法      ∴$A1＝－95
```

```
    %  10100001……$A1
          ↓反転
    %  01011110
    %         1……1を足す
    %  01011111＝$5F＝95
                     ↓
                 ∴$A1＝－95
  ②反転して1を足す方法
```

```
                ↓─右からみて最初の1
    %  1010000 1……$A1
        反転   │そのまま
               ↓
    %  01011111……$5F＝95
                        ↓
  ③反転する方法        ∴$A1＝－95
```

図４－36　補数の見方

となるような数βのことをいいます。すなわちαに対する2^8の補数βは

$\beta = 2^8 - \alpha$

で与えられるわけです。ですから、この方法はまさに定義どおりに補数を求める方法です。

☆方法2☆

与えられた数の各ビットを反転させた（0を1に、1を0にする）後で、1を加える方法です(図4—36②)。

☆方法3☆

与えられた数を下位のビットからみていき、最初に出てくる1まではそのままにして、それより上の桁のビットを反転する方法です(図4—36③)。

以上3種類の方法があります。図4—37に若干の例をあげておきます。さらに、負の数から補数表現を得る方法も、全く同じ方法で可能であることも補数の特徴です。この例も図4—38にあげておきます。

ここでは主に8ビットでの補数を扱いましたが16ビットの数の場合でも方法1で2^8が2^{16}に変わる以外は方法は同じです。

この補数は少し難しい概念かもしれません。わからない場合にはとばして、後程読み返してもけっこうです。

```
①$FFはいくつか（方法1）
    %100000000
－) %  11111111……$FF
       00000001……$01＝1
                ∴$FF＝－1

②$88はいくつか（方法2）
    %  10001000……$88
          ⇓反転
    %  01110111
＋            1
       01111000……$78＝120
                ∴$88＝－120

③$B8はいくつか（方法3）
           ↓―最初の1
    %  1101 1000……$B8
       反転 そのまま
    %  00101000……$28＝40
                ∴$B8＝－40
```

図4－37　補数表現→負の数

```
①－40を方法3で求める
                       ↓―右から見た最初の1
   40d＝$28＝%  0010 1000
                反転 そのまま

－40d＝$B8＝%  11011000

②－1を方法1で求める
              %100000000
        1d＝%  00000001
 －1＝$FF＝%  11111111

③－1234を方法3で求める
     1234＝$04D2
          ＝%00000100110100 10
                 反転       そのまま
  －1234＝%1111101100101110
          ＝$FB2E
```

図4－38　負数→補数表現

10. 補数の性質

前節で、補数表現（正確には2^8の補数表現）を用いることにより、1バイト（8ビット）ならば

－128（$80）～＋127（$7F）

の数を、2バイト（16ビット）ならば

－32768（$8000）～32767（$7FFF）

の数を表現できることがわかりました。

しかしなぜこのように複雑な補数を用いるので

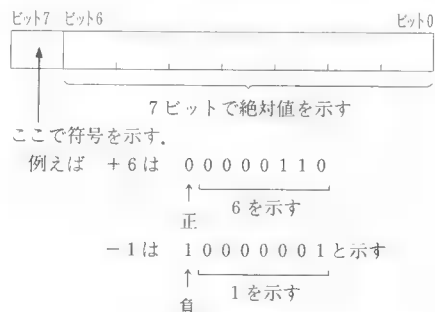



図4－39　補数を用いない負の表現

しょうか。例えばもっと簡単に図4―39のように負数を表現しないのでしょうか。

実は、補数には非常に便利な性質を持っているのです。これまでは正の数と正の数の加算を扱いましたが、この**補数を用いれば正の数と負の数の混合した加算が、今までの加算とほとんど同じ方法で求めることができる**のです。例をあげましょう。図4―40の例をみてください。それぞれ負の数を含んだ計算をしています。例1と例3でビット7からの繰りあがりが生じていますが、この繰りあがりは無視することにします。すると負の数を含んだ加算は、正の数同志の加算と区別する必要がなくなり、たいへん便利です。ちなみに補数を用いない図4―39の表現法で加算を行った例を図4―41に示しておきます。この場合は正確な値が得られません。もし正確な値を求めたければ、正＋正、正＋負、負＋負のそれぞれに専用の加算命令が必要になり、非常に不便です。

11. Vフラグ

さて、補数表現を用いると負の数を表現することができるのですが、表現できる数の範囲が定まっています（1バイトならば－128～＋127）。

また、絶対値表現を用いた場合にも表現できる数の範囲は決っています（1バイトならば、0～255）。

このため、演算を行った結果が上述の範囲外になった場合には、正しい答は得られません。ＣＰＵは、このように正しい答とならない場合には、ＣＣレジスタ内の各フラグを変化させて、私たちに知らせてくれます。

まず絶対値表現とみなしている場合は、演算の

〔例1〕　6＋(－4)＝＋2
（今までの加算と同じ）

```
   00000110……… 6
+  11111100……－4
  100000010……… 2
  ↑
  無視
```

〔例2〕　(－6)＋4＝－2

```
   11111010……－6
+  00000100……… 4
   11111110……－2
```

〔例3〕　(－6)＋(－4)＝－10

```
   11111010……－6
+  11111100……－4
  111110110…－10
  ↑
  無視
```

図4－40　負の数を含んだ加算

〔例〕　6＋(－4)＝2の計算

```
   ┌符号
   ↓
   00000110…………6
+  10000100………－4
   10001010……－10 ←間違った答が出る
   ↓└──────┘
   負    10
```

図4－41　補数を用いない負の表現の欠点

絶対値表現	CPUの計算	補数表現
5	0000 0101	5
＋	＋	＋
200	1100 1000	－56
＝	＝	＝
205	1100 1101	－51
	C＝0　V＝0	
OK←		→OK
100	0110 0100	100
＋	＋	＋
200	1100 1000	－56
＝	＝	＝
44（×）	0010 1100	44
	C＝1　V＝0	
BAD←		→OK
100	0110 0100	100
＋	＋	＋
44	0010 1100	44
＝	＝	＝
144	1001 0000	－112（×）
	C＝0　V＝1	
OK←		→BAD
144	1001 0000	－112
＋	＋	＋
205	1100 1101	－51
＝	＝	＝
93（×）	0101 1101	93（×）
	C＝1　V＝1	
BAD←		→BAD

図4－42　CフラグとVフラグの変化

結果が範囲を超えた場合にはCCレジスタ内のCフラグがセットされ、超えなかった場合にはリセットされます。Cフラグがセットされるのは、加算で最大数（1バイトの時は255）を超えた場合と、減算で答えが負になる場合です。

次に補数表現とみている場合に、演算の結果が範囲を越えた場合にはCCレジスタ内のVフラグ（オーバーフローフラグ）がセットされ、超えなかった場合にはリセットされます。

例を図4—42に示します。CPUは補数表現なのか、絶対値表現なのかは知りませんので、CフラグとVフラグの両方を変化させます。その変化を人間がどちらの表現を用いているかによって、人間の側でCフラグを用いるか、Vフラグを用いるかを決定するわけです。

12. CCレジスタ

ここではCCレジスタ（Condition Code Register）について解説します。

このCCレジスタはみかけ上は8ビットのレジスタですが、その特徴からいって、1ビットのレジスタが8個あると考えた方がよいでしょう。(図4—43)。CCレジスタの各ビットは、CPUの演算の結果で生じた付加情報を記憶しておく役目を持っており、各ビットに意味と名前があります（E，F，Iの各フラグについては実践編で解説します)。

Cフラグは、既に述べたように絶対値表現の範囲を超えた場合にセットされるフラグです。

Vフラグは、補数表現の範囲を超えた場合にセットされます。

Zフラグは、演算の結果が0であるときにセットされます。

Nフラグは、演算の結果を補数表現とみなした場合に、結果が負の数である場合にセットされます。

Hフラグは、ほとんど用いられませんが、加算命令を実行したときにビット3からビット4への繰りあがりがあるときにセットされます。

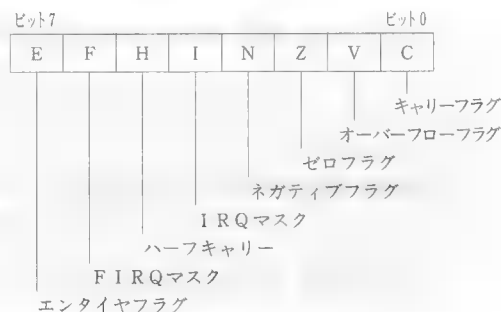
これらのフラグは演算の結果、それぞれの条件が成立したときにセットされ(1になる)、成立しないときにはリセットされ（0になる）ます。

つまり、CCレジスタの各フラグは、演算の結果生じた付加情報を我々に提供しているわけです。ここで注意しなければならないのは、CCレジスタの各フラグは、どの命令でも変化するわけではなく、命令の種類によって変化するフラグと変化しないフラグがあるということです。

付録のインストラクション表をみてください。各命令の1番右に「HNZVC」の欄があります。ここに、各命令でどのフラグが変化するかが示してあります。LD命令を例にみてみましょう。

まずHフラグとCフラグの欄に「↩」の記号があります。これはこの命令（ここではLD命令）を実行してもこのフラグは変化せず、前の状態を保存するということを示しています（図4—43)。

次にNフラグとZフラグの欄に「↕」の記号があります。これは、この命令の実行結果によって各フラグの設定条件を満たす場合には1（セット)、条件を満足しない場合には0（リセット）になることを示しています。

Vフラグの所の「0」は、この命令を実行することにより必ず0になることを示しています。ま

CCレジスタ（名前と略号）

インストラクション	ニーモニック	OP	5 H	3 N	2 Z	1 V	0 C
LD	LDA	86	●	↕	↕	0	●
	LDB	C6	●	↕	↕	0	●
	⋮		⋮	⋮	⋮	⋮	⋮

図4－43　インストラクション表（一部）

た、LD命令には出てきませんが「1」はこの命令を実行することにより必ず1になることを示しています。0、1以外の数字は注釈の番号を示しているので末尾の注釈を参照してください。

文章だけでは理解しにくいと思いますので、いろいろな例を図4—44に示します。

13. 8ビットと16ビットの演算

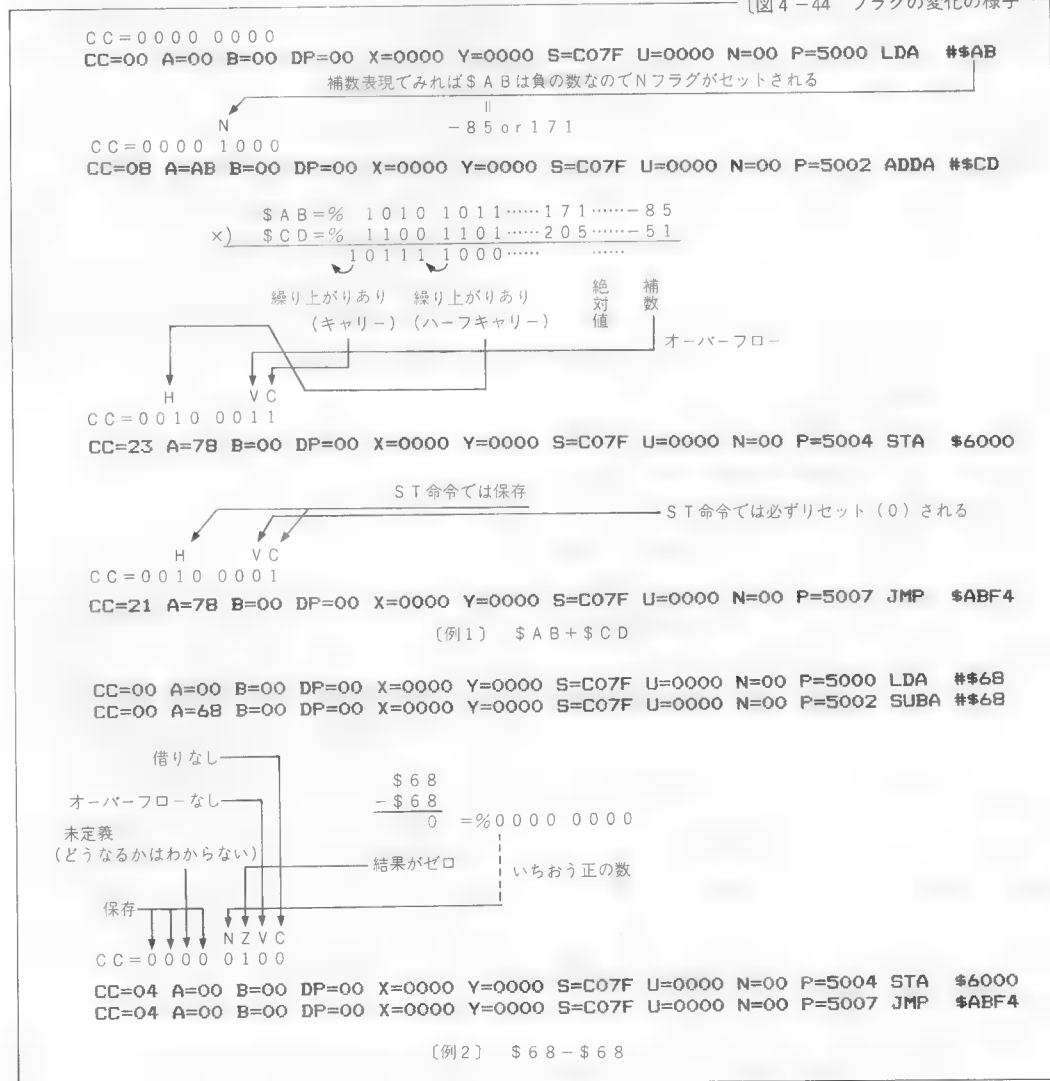
ここまでの説明で加算と減算の方法と仕組みについては理解できたと思います。ここではその変形を扱います。今までは、演算を行う場合には同じ長さ（1バイトなら1バイト）のもの同志で行ってきました。しかし、ときには、8ビットの数と16ビットの数を加算しなければならないこともあります。次の課題をみてください。

課題3

＄6000番地に格納されている数の符号を反転させてから、＄6001、＄6002番地の内容を加えて、得られた結果を＄6003、＄6004番地に格納せよ。

〔図4－44　フラグの変化の様子

〔例1〕　＄AB＋＄CD

〔例2〕　＄68－＄68

まずは課題を分析しながらプログラムを考えていきましょう。

最初に＄６０００番地に格納されている数の符号を反転させる、つまり、正負の逆転させるわけです。この動作をするには NEG 命令を用います。この NEG命令は、指定したアドレスか指定したレジスタの内容の２の補数を取り、または元のところへ戻す命令です。２の補数を取ることは、正負を逆転させることですから、この命令でよいわけです。NEG 命令には、エクステンドモードの他に、

NEGA　（Aレジスタの内容の補数を取る）
NEGB　（Bレジスタの内容の補数を取る）

Aレジスタ　０１０１１０１０　＝９０ｄ
＄５A
NEGA
Aレジスタ　１０１００１１０　＝－９０ｄ
＄A６

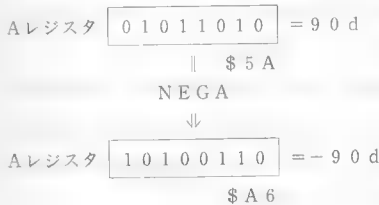



図４－45　ＮＥＧＡ命令

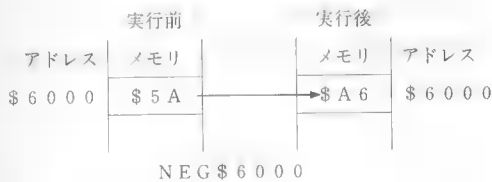



図４－46　ＮＥＧ命令

などの命令があります。この２つのようにオペランドを必要としないものをインヘレントモードと

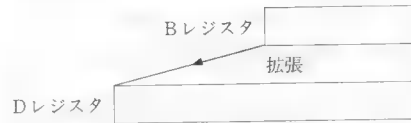



図４－47　８ビットから16ビットへ拡張

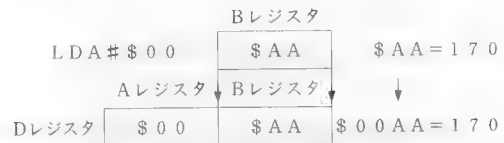



図４－48　絶対値表現のときの拡張

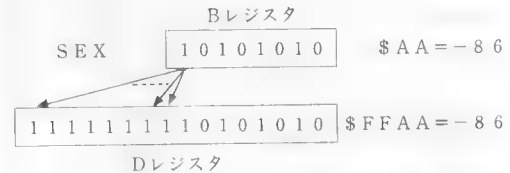



図４－49　補数表現のときの拡張（負数の場合）

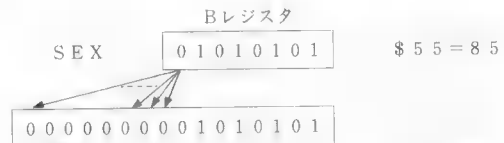



図４－50　補数表現のときの拡張（正数の場合）

［図４－51　課題３のプログラム］

```
5000                    ORG   $5000
5000 70    6000         NEG   $6000 ……………… 符号反転
5003 F6    6000         LDB   $6000
5006 1D                 SEX
5007 F3    6001         ADDD  $6001 ……………… 加算
500A FD    6003         STD   $6003 ……………… 格納
500D 7E    ABF4         JMP   $ABF4
                        END

0 ERROR(S) DETECTED

SYMBOL TABLE:
```

いいます。

本題に戻って、この場合には、＄６０００番地の符号を反転させたいのですから、NEG 命令をエクステンドモードで用いればよいわけです。

次に、まず＄６０００番地の内容をＢレジスタにロードします。ここで8ビットから16ビットへ、すなわちＢレジスタからＤレジスタへ数を拡張しなければいけません。というのは、演算は特別な場合を除いて同じ長さのデータどうしで行わなければいけないからです。

Ｂレジスタの内容が絶対値表現の場合には、上位＝Ａレジスタを0にするだけでよいので

LDA #$00

で十分です（他により適した命令もあるのですがここでは、これでいいことにします)。

しかし、補数表現となると簡単にはいきません。補数表現を用いている場合には、符号部（ビット７）を正確に拡張しなければならないからです。この動作を行うのが SEX 命令です。この命令は、**Ｂレジスタに格納されている数を正確に拡張してＤレジスタに格納します。**詳しい動作は図4―49、図4―50をみれば一目瞭然ですが、単にＢレジスタのビット７をＡレジスタの全てのビットにコピーするわけです。

これでＤレジスタに＄６０００番地の（反転した後の）内容が16ビットに拡張されて格納されました。後は、＄６００１、＄６００２番地の内容をＤレジスタに加算して、＄６００３、＄６００４番地に格納すれば終了です。

以上の分析に基づいて作成されたのが図4―51です。課題１どおりの動作をするかどうか各自で十分に確かめてください。実行例を図4―52、さらに実行の様子を図4―53に示します。

［図４－52　課題３の実行］

```
*D5000

5000 70 60 00 F6 60 00 1D F3
5008 60 01 FD 60 03 7E AB F4
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

［ダンプリスト］

```
*M6000
6000 00-10 ……………… $10をセット
6001 00-12 ┐
6002 00-34 ┘ ……………… $1234をセット
6003 00-
6004 00-.

*G5000 ……………… 実行

*M6000 ……………… －$10＝$F0になっている
6000 F0-
6001 12- ┐
6002 34- ┘ ……………… $1234
6003 12- ┐
6004 24-.┘ ……………… $1224＝（－$10）＋$1234
```

14. 乗算

本章の最後に乗算を行う MUL 命令をあげておきましょう。

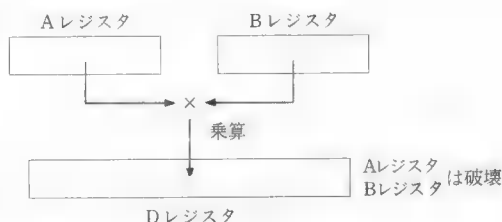



図４－54　ＭＵＬ命令

［図４－53　ＳＥＸ命令の実行の様子］

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 NEG  $6000
CC=09 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 LDB  $6000
CC=09 A=00 B=F0 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5006 SEX……符号拡張
CC=09 A=FF B=F0 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 ADDD $6001
CC=01 A=12 B=24 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500A STD  $6003
CC=01 A=12 B=24 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500D JMP  $ABF4
REF-MP TRAP AT 500D
**
```

このMUL命令は、Aレジスタの内容×Bレジスタの内容を計算し、得られた結果をDレジスタに格納します。ただし、ここで各レジスタの内容は絶対値表現のものに限定されていて、補数表現の場合には正確な答は得られません。図示すると図4—54の動作をします。

結果が格納されるDレジスタは、AレジスタとBレジスタの別名ですから当然AレジスタとBレジスタの元の値は破壊されて失われてしまいます。

この便利な（実は乗算の命令を持っている8ビットのCPUは少いのです）MUL命令を用いたサンプルを図4—55、実行例を図4—56に示します。

最後にこの命令を使用する際の注意点をあげておきましょう。1つは、このMUL命令は、今までの演算命令（ADDなど）と違ってオペランドを必要としません。つまり、レジスタ対メモリ型の命令ではなく、レジスタ対レジスタ型の命令であるわけです。例をあげれば

MUL $6000

などという命令はないわけです。

2つめは、この命令を実行した後のCフラグの状態です。インストラクション表の注釈にもあるように、結果のBレジスタ（下位）のビット7を同じ値を取ります。これは、乗算の後に結果を8ビットに丸めたいときに使用するためのものです。

MUL

ADCA #0

この実行により乗算の近似値がAレジスタに入るわけです。ほとんど使いませんが御参考までに(まずおぼえる必要もないでしょう)。

［図4—56 MUL命令サンプルの実行］

```
*D5000

5000 B6 60 00 F6 60 01 3D FD
5008 60 02 7E AB F4 00 00 00
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

［ダンプリスト］

```
*M6000
6000 00-05     5×3をセット
6001 00-03
6002 00-
6003 00-.

*G5000

*M6000
6000 05-  ┐ 5×3=15=$0F
6001 03-  ┘
6002 00-  ┐
6003 0F-. ┘

*■
```

［図4—55 MUL命令のサンプル］

```
5000                ORG   $5000
5000 B6    6000     LDA   $6000
5003 F6    6001     LDB   $6001
5006 3D             MUL
5007 FD    6002     STD   $6002
500A 7E    ABF4     JMP   $ABF4
                    END

0 ERROR(S) DETECTED
```

［図4—57 MUL命令の実行の様子］

```
CC=01 A=12 B=24 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  $6000
CC=01 A=05 B=24 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 LDB  $6001
CC=01 A=05 B=03 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5006 MUL
CC=00 A=00 B=0F DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 STD  $6002
CC=00 A=00 B=0F DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500A JMP  $ABF4
REF-MP TRAP AT 500A
**
```

第 5 章

マシン語プログラミング
part III

——比較とループ——

1. CMP命令

前章までで計算機の大きな2つの役割のうち、計算の部分を学習してきました。この章ではもう1つの役割である繰り返して仕事をこなすというところに重点を置いて考えます。

つまり、BASICではGOTO文やIF文を利用して行っている分岐と判断（比較）という動作をマシン語で実現しようというわけです。

課題4

＄6000番地に格納されている値、(絶対値表現）を参照して

①その数が60以上のときには＄6001番地に1を格納する。

②その数が60未満のときには＄6001番地に0を格納する。

この課題を実現するには、比較と分岐を実現する命令を学習しなければいけません。そこでまず比較命令を学習しましょう。

そこでAレジスタの値と60とを比較することを考えてみます。これだけならば実は既に学習した命令で可能なのです。その命令は SUB 命令です。この場合、

```
SUBA    #60
```

とすれば、Aレジスタが60未満のときには、引くことができないので借りが生じてCフラグがセットされます。Aレジスタが60以上であれば、単純に引くことができ、借りは生じないのでCフラグはリセットされます。また、 SUB命令ではZフ

Aレジスタ	比較の基準
X	Y

X－Yを実行
- ① 引ける場合にはCフラグがリセットされる（C＝0）………X≧Y
- ② 引けない場合にはCフラグがセットされる（C＝1）………X＜Y

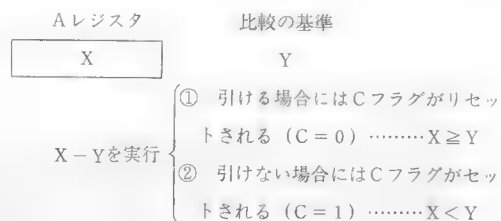



図5－1 比較の方法

ラグも設定されるので、Aレジスタがちょうど60だったならば、Zフラグがセットされます。

このように、Aレジスタを何かの値（今の場合には60という定数でしたが、エクステンドモードを用いれば、＄６００２番地の内容と比較するということもできます）と比較するには、実際に引いてみるのが最適というわけです。

しかしSUB命令を使用することには１つだけ欠点があります。それはSUB命令で比較するとAレジスタの内容は引き算の結果と入れ換ってしまい、元のAレジスタの内容は失われてしまいます。これを避けるには、Aレジスタの内容をいったんどこかに退避させておいて、後で元のところに戻してやればよいのですが、いちいちそのために退避場所を設定しなければならないため不便です。そこでCMP命令の登場というわけです。

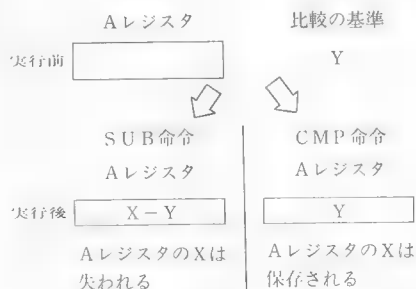
CMP命令は、指定したレジスタから指定した値を引き、その結果によってCCレジスタの各フラグを変化させます。ここまではSUB命令と変わりません。この後SUB命令では結果をAレジスタに再格納していたわけですが、CMP命令は結果を再格納しません。ここだけが違います。

これを使えばAレジスタの内容を置き換えずに比較することができ、たいへん便利です。課題の場合には

```
CMPA    #60
```

でよいわけです。

では、この命令を実行すると各フラグはどう変化するのでしょうか。この命令はインストラクション表にあるとおり、N、Z、V、Cのフラグが

図5－２　CMPとSUBの違い

変化します。説明のために、指定したレジスタの内容をX、比較の基準になる値（指定した値）をYと置くことにします。

Zフラグは、減算の結果が０になったときセットされるのですから、実行後Zフラグがセットされている（１となっている）ならば、X＝Yであったことがわかります。つまり

Z＝１⟶X＝Y

Z＝０⟶X≠Y

ということです。

Cフラグは、レジスタの内容と、比較の基準値を絶対値表現とみなして減算したときの借りの有無を示します。

C＝１⟶借りあり⟶X＜Y

C＝０⟶借りなし⟶X≧Y

ZフラグやVフラグもある規則によって変化しますが、複雑になるので省略します。省略しても構わないわけは後程解説します。

2. 条件付ブランチ命令(1)

さて、前節では比較の方法を述べました。比較することによってCCレジスタにさまざまな情報がセットされます。次にこの情報の利用の方法に入りましょう。

私たちはBASICでは、

```
IF  A>B  THEN  230
```

などのように、比較の後に必ず、プログラムの枝分かれがありました。マシン語でも同じで、比較のCMP命令の後ろには、多くの場合プログラムの枝分かれがあります。すなわち分岐命令（ブランチ命令）が置かれているわけです。

ではブランチ命令にはどのような命令があるのでしょうか。図5—3をみてください。これらがブランチ命令です。これらはそれぞれの条件が満たされるときだけ分岐します。もし条件が満たされなければ何もしません。

さて、ここで課題に戻りましょう。まず各自で課題を分析して、その処理の過程のイメージを描いてください。BASICでもフローチャートでも結構です。人によりさまざまだと思いますが、

各自バラバラでは話にならないので、図5—4に例をあげます。これを見ると、もうプログラムの大部分が完成しているのがわかると思います。あと命令が書かれていないのは、分岐するところだけです。いよいよその命令を数ある分岐命令の中から選ぶことになります。

図5—5をみてください。これは数ある分岐命令を系統的に分類したものです。これを用いてどの分岐命令を選択するのかを考えていきます。

まず、＄６０００番地に格納してある値や60という定数が符号付きであるか符号なしであるかを確認します。符号付き、符号なしというのはそれぞれ、補数表現と絶対値表現というのと同じです。課題では絶対値表現と断わってあるので、符号なしの方を選びます。

次に分岐する条件を選びます。図5—4にもあるように「A＜60」のときに分岐すればよいわけです。Testの項をみると、「ｒ＜ｍ」というのがあります。ここで「ｒ」というのは、分岐命令の前に比較したときのレジスタの内容を示し、「ｍ」というのは、比較したときの基準値を示しています。ですから

「A＜60」＝「ｒ＜ｍ」

ここでは条件「A＜60」が成立したときに分岐してほしいのですから「True」の項の「BLO」を選びます。「False」の項は左の条件が不成立の際に分岐したい場合に使用します。

さて、ここでCMP命令のところでフラグの変化を詳しく解説しなかった理由を述べましょう。本来条件付分岐命令は、その時のCCレジスタの各フラグの状態をみて、それ自身の分岐条件と検討をして分岐するか、しないかを決定します。例えば、図5—5の中にあるBGE命令は、CCレジスタ中のフラグを参照して「N＝１でかつV＝１である場合と、N＝０でかつV＝０の場合」に分岐すると定められています。一方、CMP命令では、レジスタの内容をｒ、基準値をｍとすれば、「ｒ≧ｍ」のときに、「N＝１かつV＝１か、N＝

インストラクション	ニーモニック	Op	~#	#	動作	5 H	3 N	2 Z	1 V	0 C
BCC	BCC	24	3	2	Branch C=0	●	●	●	●	●
	LBCC	10	5(6)	4	Long Branch	●	●	●	●	●
		24			C=0					
BCS	BCS	25	3	2	Branch C=1	●	●	●	●	●
	LBCS	10	5(6)	4	Long Branch	●	●	●	●	●
		25			C=1					
BEQ	BEQ	27	3	2	Branch Z=1	●	●	●	●	●
	LBEQ	10	5(6)	4	Long Branch	●	●	●	●	●
		27			Z=1					
BGE	BGE	2C	3	2	Branch≧Zero	●	●	●	●	●
	LBGE	10	5(6)	4	Long Branch≧Zero	●	●	●	●	●
		2C								
BGT	BGT	2E	3	2	Branch＞Zero	●	●	●	●	●
	LBGT	10	5(6)	4	Long Branch＞Zero	●	●	●	●	●
		2E								
BHI	BHI	22	3	2	Branch Higher	●	●	●	●	●
	LBHI	10	5(6)	4	Long Branch Higher	●	●	●	●	●
		22								
BHS	BHS	24	3	2	Branch Higher or Same	●	●	●	●	●
	LBHS	10	5(6)	4	Long Branch Higher or Same	●	●	●	●	●
		24								
BLE	BLE	2F	3	2	Branch≦Zero	●	●	●	●	●
	LBLE	10	5(6)	4	Long Branch≦Zero	●	●	●	●	●
		2F								
BLO	BLO	25	3	2	Branch lower	●	●	●	●	●
	LBLO	10	5(6)	4	Long Branch Lower	●	●	●	●	●
		25								

インストラクション	ニーモニック	アドレッシングモード リラティブ Op	~	##	動作	5 H	3 N	2 Z	1 V	0 C
BLS	BLS	23	3	2	Branch Lower or Same	●	●	●	●	●
	LBLS	10	5(6)	4	Long Branch Lower or Same	●	●	●	●	●
		23								
BLT	BLT	2D	3	2	Branch＜Zero	●	●	●	●	●
	LBLT	10	5(6)	4	Long Branch＜Zero	●	●	●	●	●
		2D								
BMI	BMI	2B	3	2	Branch Minus	●	●	●	●	●
	LBMI	10	5(6)	4	Long Branch Minus	●	●	●	●	●
		2B								
BNE	BNE	26	3	2	Branch Z=0	●	●	●	●	●
	LBNE	10	5(6)	4	Long Branch	●	●	●	●	●
		26			Z=0					
BPL	BPL	2A	2	2	Branch Plus	●	●	●	●	●
	LBPL	10	5(6)	4	Long Branch Plus	●	●	●	●	●
		2A								
BRA	BRA	20	3	2	Branch Ahways	●	●	●	●	●
	LBRA	16	5	4	Long Branch Always	●	●	●	●	●
BRN	BRN	21	3	2	Branch Never	●	●	●	●	●
	LBRN	10	5	4	Long Branch Never	●	●	●	●	●
		21								
BSR	BSR	8D	7	2	Branch to Subroutine	●	●	●	●	●
	LBSR	17	9	3	Long Branch to Subroutine	●	●	●	●	●
BVC	BVC	28	3	2	Branch V=0	●	●	●	●	●
	LBVC	10	5(6)	4	Long Branch	●	●	●	●	●
		28			V=0					
BVS	BVS	29	3	2	Branch V=1	●	●	●	●	●
	LBVS	10	5(6)	4	Long Branch	●	●	●	●	●
		29			V=1					

図５－３　ブランチ命令

0かつV＝0」というぐあいにフラグが変化します。

通常CMP命令で比較した後には分岐が伴います。つまりペアで使用されるのがほとんどです。ですから、「N＝1かつV＝1か、N＝0かつV＝0」などという条件は抜きにして、CMP命令のあとに、BGE命令を置けば、「r≧m」のときに分岐すると覚えておけば十分ということになります。

幸い、例にあげた「BGE」は「Branch on Greater

図5－4　フローチャート

ＣＭＰ命令

「r≧m」ならば　「N＝1かつV＝1」 または 「N＝0かつV＝0」

ＢＧＥ命令

「N＝1かつV＝1」 または 「N＝0かつV＝0」 ならば分岐する

⇩ 三段論法で

ＣＭＰ命令＋ＢＧＥ命令

「r≧m」ならば分岐する

図5－6　フラグの変化は不要

ブランチ

	OP	~	#
BRA	20	3	2
LBRA	16	5	3
BRN	21	3	2
LBRN	1021	5	4
BSR	8D	7	2
LBSR	17	9	3

符号付きブランチ

Test	True	OP	False	OP
r>m	BGT	2E	BLE	2F
r≧m	BGE	2C	BLT	2D
r=m	BEQ	27	BNE	26
r≦m	BLE	2F	BGT	2E
r<m	BLT	2D	BGE	2C

条件付きブランチ

Test	True	OP	False	OP
N=1	BMI	2B	BPL	2A
Z=1	BEQ	27	BNE	26
V=1	BVS	29	BVC	28
C=1	BCS	25	BCC	24

符号なしブランチ

Test	True	OP	False	OP
r>m	BHI	22	BLS	23
r≧m	BHS	24	BLO	25
r=m	BEQ	27	BNE	26
r≦m	BLS	23	BHI	22
r<m	BLO	25	BHS	24

図5－5　ブランチ命令

than or Equal to Zero」の略で、意訳すれば「比較した結果が、ゼロと等しいか、ゼロより大きい時にブランチ（分岐）する」という意味ですから、CMP命令が「r－m」で比較することとあわせれば

r－m≧0　　∴　r≧m

のときに分岐する命令であることを示しています（他の命令のフルネームは後述します）。

3. 条件付ブランチ命令(2)

前節でレジスタの基準値からの大小による分岐命令について述べてきました。そして、こういった場合、各フラグの変化を覚える必要がないことを示しました。

しかし、プログラムによっては、そのときのフラグのセット，リセットの状態で分岐する命令も必要となります。例えば、Nフラグが1のとき分岐したいなどという場合です。こういった要求には図5―7の命令が応えてくれます。

Nフラグが1のときに分岐したい場合は、“BMI”命令を使用します。どういう場合に用いるのかというと、＄６０００番地に格納されている値が負のとき分岐したいとして、

```
LDA $6000
BMI ?????
```

で分岐できるわけです。これは、LDA命令によってNフラグが設定されることを利用したものです（？の部分は次節で解説します）。

他にも、加算の結果、繰りあがりが生じた際に分岐したい場合

```
ADDA #$60
BCS ?????
```

で、実現できます。もしADDA命令で繰りあがりが生じれば、Cフラグがセットされるので、BCSで分岐し、繰りあがりが生じていなければCフラグはリセットされてBCS命令では分岐せずにそのまま次の命令に実行が移るということです。

さて、こういった調子に条件付分岐命令には、CMP命令の後に用いて比較の結果の大小に応じて分岐する命令群（前節で説明したもの）と、フラグのセット，リセットの状態で分岐する命令群（この節で説明したもの）とがあります。しかしこのように分類すると、名前が違っても同じ命令というものも存在します。例をあげると、BLO命令は絶対値表現で比較の結果、r＜mのときに分岐する命令ですが、その実態はCフラグがセットされている時に分岐する命令ですから、BCS命令と違いがありません。これは、あくまで人間にわかりやすくするためです。

ここで、もう1種だけ分岐命令を紹介しておきましょう。ここまでは、ある条件が満たされたときに分岐する命令を調べてきましたが、**無条件で分岐する命令**もあります。BRA命令がその命令です。この命令は、CCレジスタがどうであろう

条件付きブランチ				
Test	True	OP	False	OP
N＝1	BMI	2B	BPL	2A
Z＝1	BEQ	27	BNE	26
V＝1	BVS	29	BVC	28
C＝1	BCS	25	BCC	24

図5―7　条件付きブランチ

				OPコード
BLO	＝	BCS	＝	＄25
BHS	＝	BCC	＝	＄24

図5－8　名前は違うが同じ命令

```
$5000  LDA   $6000
       CMPA  #60
       BLO
       LDA   #$01
       BRA
       LDA   #$00
       STA   $6001
       JMP   $ABF4
```

図5－9　課題4のプログラム（簡略図）

と、必ず分岐する命令です。図5—4をもう一度みてください。この図では、Aレジスタに1を代入した後、処理の順序を示す矢印が大きく迂回しています。もしこの迂回がなければ、そのまま下へ移り、Aレジスタに0を代入してしまいます。この様に処理を迂回させるのに、用いるのがBRA命令です。

さて、これで課題4のプログラムがだいぶ完成に近付いてきました。図5—9は簡略化して示しています。

4. リラティブモード

前節までで各種の分岐命令がでてきましたが、その分岐命令の分岐先はどう指定したらよいのでしょうか。今までの各種の命令はオペランドでその命令の演算対象を示していました。分岐命令も同様に分岐先をオペランドで示します。オペランドといえばアドレッシングモードですが、分岐命令にはどんなアドレッシングモードがあるのかをみてみましょう。図5—10を参照してください。これは付録のインストラクション表の一部です。各種の分岐命令が並んでいますが、アドレッシングモードは、リラティブモードしかありません。つまり、分岐命令では必ずリラティブモードを用いるわけです。

それでは、分岐先はどう表記したらよいかというと、表記上はエクステンドモードと同じくなにも付けずに、アドレスを書きます。例えば、＄500B番地に分岐したいのであれば

BLO　$500B

と記述します。このままですと、エクステンドモードと混乱しそうですが、リラティブモードは分岐命令にしか存在しないので他と混乱することはありません。

では、

BLO　$500B

をハンドアセンブルするにはどうしたら良いのでしょう。ここで、リラティブモードのハンドアセンブルの仕方を学習しましょう。

例として課題4の場合を考えてみます。図5—9をできる限りハンドアセンブルしたものが図5—11です。分岐命令は、後でみますが2バイトにアセンブルされるので、その分を空けておきます。

ここまで書けば、分岐命令の分岐先は決定したことになります。それぞれ、

BLO　$500B
BRA　$500B

となります。

まずOPコードは、付録のインストラクション表を参照することにより、今までと同じ方法で求められて

BLO ⟶　＄25
BRA ⟶　＄20

とわかります。次にオペランドの設定です。このオペランドの値は次のようにして求められます。

分岐命令のOPコードを格納したアドレスをa番地、分岐先のアドレス（アセンブリ表記でオペランドに書いたアドレス）をb番地、この分岐命令全体の命令の長さをcバイトとすると、求めるオペランドの値xは

$x = b - a - c$

で与えられます。

実際の例でみてみましょう。まずBLOの方は、OPコードのアドレスは＄5005、分岐先は＄500Bです。次に命令長ですがインストラクション表をみます。インストラクション表の各命令の部分には「OP　～　#」と三つの欄があります。「OP」は既に述べているOPコードですが「#」の欄が命令長（命令の全体のバイト数）を表しています。BLOの項をみると命令長は2バイトで

インストラクション	ニーモニック	アドレッシングモード リラティブ Op	～	##	動作	5 H	3 N	2 Z	1 V	0 C
BLS	BLS	23	3	2	Branch Lower or Same	●	●	●	●	●
	LBLS	10 23	5(6)	4	Long Branch Lower or Same	●	●	●	●	●
BLT	BLT	2D	3	2	Branch＜Zero	●	●	●	●	●
	LBLT	10 2D	5(6)	4	Long Branch＜Zero	●	●	●	●	●
BMI	BMI	2B	3	2	Branch Minus	●	●	●	●	●
	LBMI	10 2B	5(6)	4	Long Branch Minus	●	●	●	●	●

図5－10　ブランチ命令表（一部）

す。ですから求めるオペランドの値xは

x＝＄５００B－＄５００５－２
＝６

を補数表現したものとなります。

同様にBRAの方は、

x＝＄５００D－＄５００９－２
＝２

となります。ですから、完成したプログラムを表したものが図5―13です。アセンブラを用いれば、複雑なオペランドの計算を実行してくれますので、非常に便利です。でも便利だからといって頼ってばかりではいけません。複雑な例を１つあげておきます。次の命令をハンドアセンブルしてみます。

$5000 BRA $5000

これは、無限ループになっています。このときのオペランドの値xは

x＝＄５０００－＄５０００－２
＝－２

となります。－２を補数表現すると＄FEとなるので、OPコードとあわせて

＄２０　＄FE

となるわけです。

１バイトの補数表現で表わせる数は、

－１２８～＋１２７

ですから、前にも後にも分岐できますが、その範囲は限られることになります。

5. アセンブラのラベル

さて、ここでもう一度図5―9に戻ってみます。この時点では、分岐命令の分岐先のアドレスはわかっていません。次に図5―11をみてください。このときにやっと分岐先のアドレスがわかりました。しかし、このときになってやっとわかったのでは、わざわざアセンブラを使う気にもなりません。というのは、分岐先のアドレスを知るには、１度分岐命令以外の全命令をハンドアセンブルしなければならなかったからです。もう、ほとんどハンドアセンブルし終っているのですから、いまさらアセンブラを使う必要もないわけです。こんな非能率的なことがあっていいはずがありません。

実は、アセンブラには、(非常に便利な)ラベルという概念が使えるのです。ラベルというのは分岐先に名前をつけて用います。ラベルを用いた例を図5―14に示します。つまり図5―9において矢印で示したものを対応する名前＝ラベルで示すことができるわけです。このようにしておくとアセンブラは、ラベルの値(ここで“ HIKUI ”は＄５００Bとなる）を求めて、きちんとアセンブルしてくれます。

6. ロングブランチ命令

前々節で条件付きブランチ命令（分岐命令）は、ある限定された範囲へしか分岐できないことを示しました。6809の先祖である6800では、この制限がついてまわり、長いプログラムを開発するとき

５０００	B６	６０	００	LDA	＄６０００
５００３	８１	３C		CMPA	＃６０
５００５	？？	？？		BLO	
５００７	８６	０１		LDA	＃＄０１
５００９	？？	？？		BRA	
５００B	８６	００		LDA	＃＄００
５００D	B７	６０	０１	STA	＄６００１
５０１０	７E	AB	F４	JMP	＄ABF４

図５－１１　課課４のハンド・アセンブル

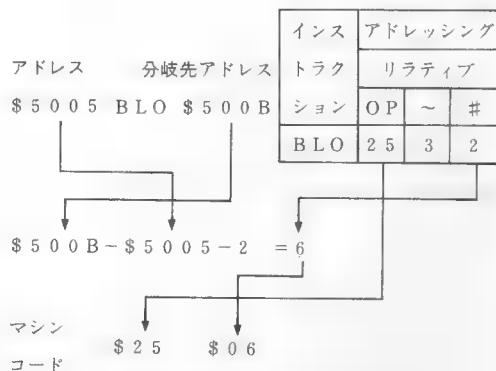



図５－１２　BLO命令のハンドアセンブル

に不便な思いをしたものでした。しかし6809ではこの欠点は克服されています。**ロングブランチ命令**が解決してくれます。このロングブランチ命令は、これまでの分岐命令が、分岐先を１バイトの補数表現で表していたものを、**２バイトの補数表現で表わしているものです。**

　ロングブランチ命令というふうに呼びますが、それほど大げさなものではありません。ニーモニックは今までの分岐命令のニーモニックの前に“L”を付けただけです。たとえばBRA命令のロング版は“ LBRA ”となるわけです。

　このロングブランチ命令を用いると

　－３２７６８～＋３２７６７

の範囲に分岐できることになり、こうなると、6809のアドレスのどこからどこにでも分岐できるということになります。

　図5―15は図5―14のプログラムで分岐命令に（別に必要もないのですが）ロングブランチ命令を用いてみたプログラムです。ロングブランチ命令を用いると、ＯＰコードが（ LBLO というように）２バイトになるものもあります。 LBLO と LBRA の行の左に“＞”のマークが出力されていますが、これはアセンブラが「ここはロングブランチ命令を使う必要はありませんよ」と教えてくれているのです（他のアセンブラでは“ＷＡＲＮＩＮＧ（警告）”と出力されるものもあります）。これは単なる注意であって間違っているわけではありませんから、実行できます。

図５－13　課題４の完成プログラム

```
5000                      ORG   $5000
5000 B6    6000           LDA   $6000
5003 81    3C             CMPA  #60
5005 25    04             BLO   $500B
5007 86    01             LDA   #$01
5009 20    02             BRA   $500D
500B 86    00             LDA   #$00
500D B7    6001           STA   $6001
5010 7E    ABF4           JMP   $ABF4
                          END

0 ERROR(S) DETECTED
```


図５－14　ラベルを使った課題４

```
5000                                ORG   $5000
5000 B6    6000                     LDA   $6000
5003 81    3C                       CMPA  #60
5005 25    04                       BLO   HIKUI
5007 86    01                       LDA   #$01
5009 20    02                       BRA   OWARI
500B 86    00             HIKUI     LDA   #$00
500D B7    6001           OWARI     STA   $6001
5010 7E    ABF4                     JMP   $ABF4
                                    END

0 ERROR(S) DETECTED
```

7. ブランチ命令の実行の様子

それでは、図5—14のプログラムを実行させて、その実行される様子を追ってみましょう。図5—17はその様子です。

まず、＄６０００番地に＄６４＝100を与えた場合の実行例（①）をみてください。CMP 命令で60と比較しましたが、100＞64ですからＣフラグはセットされません（ＣＣは０のまま）。次に分岐命令を実行します。命令名がＢＣＳになっていますがＢＬＯと同じです。これは既に述べたとおりです。ここではＣフラグはセットされていないので、そのまま次の命令へ移り、Ａレジスタに１を代入します。次にBRA 命令を実行します。これは常に分岐します。ここでＣＰＵの詳しい動きをみてみましょう。

ＣＰＵは＄５００９番地から命令を取り出して

＄２０，＄０２　（図5—14参照）

を得ます。このときＰＣ（プログラム・カウンタ）は＄５００Ｂ（次の命令のあるアドレス）を示しています。ここでＣＰＵは分岐しなければならないことを自覚します（条件つきの場合には、この時点で条件を満たすかどうかを判断します）。

分岐という動作は次の順番で行われます。まず、ＰＣにオペランドの値を加算します。すなわち

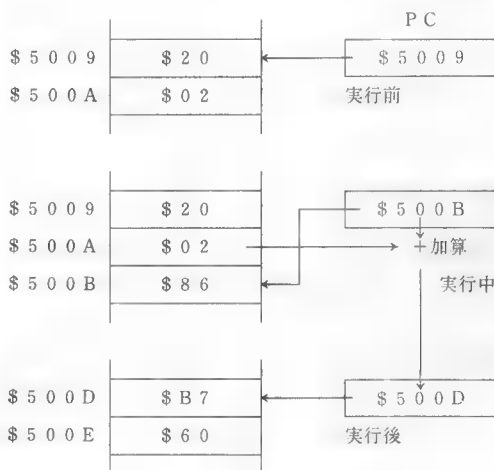



図5－16 ＢＲＡ命令の動作

図５－18　課題４のモニタによる実行例

```
*D5000

5000 B6 60 00 81 3C 25 04 86
5008 01 20 02 86 00 B7 60 01
5010 7E AB F4 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

ダンプリスト

```
*M6000 ························＄６４＝１００をセット
6000 00-64
6001 00-.

*G5000 ·········································実行

*M6000
6000 64- ················＄６４＝１００＞６０なので
6001 01-■ ·····················１がセットされている.
```

図５－15　ロングブランチ命令

```
 5000                      ORG   $5000
 5000 B6   6000            LDA   $6000
 5003 81   3C              CMPA  #60
>5005 1025 0005            LBLO  HIKUI  ┐
 5009 86   01              LDA   #$01   │ ·····················わざとロングブランチ命令を使う.
>500B 16   0002            LBRA  OWARI  ┘
 500E 86   00       HIKUI  LDA   #$00
 5010 B7   6001     OWARI  STA   $6001
 5013 7E   ABF4            JMP   $ABF4
                           END

0 ERROR(S) DETECTED
```

ＰＣ←ＰＣ＋＄０２（補数表現）
とするわけです。この例では
ＰＣ←＄５００Ｂ＋＄０２＝＄５００Ｄ
となります。これで分岐命令の動作は終了です。次にＣＰＵは命令を＄５００Ｄ番地から取り出します。すなわち
STA $6001
を実行にかかります。確かに分岐しているのがわかると思います。

これで、リラティブモードのオペランドの決定の仕方も納得できると思います。すなわち、分岐は、次の命令のアドレス（この章第４節の記号を用いれば「ａ＋ｃ」）を示しているＰＣに、オペランドの数値（同様に「ｘ」）を加算するという動作をして、分岐先のアドレス（同様に「ｂ」）を得ています。ですから「ｂ＝ｘ＋（ａ＋ｃ）」⇨「ｘ＝ｂ－ａ－ｃ」となるわけです。

次に＄３Ｃ＝60だった場合（図5—17②）には、

〔図５－17　課課４の実行の様子〕

なにもセットされない

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  $6000
CC=00 A=64 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 CMPA #$3C
CC=00 A=64 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5005 BCS  $500B
CC=00 A=64 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 LDA  #$01
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5009 BRA  $500D
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500D STA  $6001
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5010 JMP  $ABF4
REF-MP TRAP AT 5010
**
```

①　＄６４＝１００の場合

Ｚフラグセット

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  $6000
CC=00 A=3C B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 CMPA #$3C
CC=04 A=3C B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5005 BCS  $500B
CC=04 A=3C B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 LDA  #$01
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5009 BRA  $500D
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500D STA  $6001
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5010 JMP  $ABF4
REF-MP TRAP AT 5010
**
```

②　＄３Ｃ＝６０の場合

ＬＤＡ命令により，Ｚフラグ・セット

```
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  $6000
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 CMPA #$3C
CC=09 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5005 BCS  $500B
CC=09 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500B LDA  #$00
CC=05 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500D STA  $6001
CC=05 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5010 JMP  $ABF4
REF-MP TRAP AT 5010
**
```

ＣＭＰ命令により，Ｃフラグ，Ｎフラグがセットされた.（Ｚフラグはリセット）

ＬＤＡ命令により　Ｎフラグ・セット

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDA  $6000
CC=0B A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 CMPA #$3C
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5005 BCS  $500B
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 LDA  #$01
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5009 BRA  $500D
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500D STA  $6001
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5010 JMP  $ABF4
REF-MP TRAP AT 5010
**
```

ＣＭＰ命令により　Ｎフラグがセットされた

④　＄ＦＦ＝２５５の場合

CMP命令で、60−60＝0なのでZフラグがセットされています。しかし、Cフラグはリセットされているので、分岐しません。

＄００＝0だった場合（図5—17③）には、どうでしょう。CMP命令では、0−60＜0ですのでCフラグとNフラグがセットされました。Cフラグがセットされているので、分岐して＄５００B番地からに実行を移し、Aレジスタに0を代入します。

最後に、BLO命令が絶対値表現であることを確認してみたのが図5—17④です。＄FF＝２５５で試してみました。補数表現だったとすれば、＄FF＝−1で、−1＜60ですから分岐することになりますが、BLO命令は絶対値表現ですので、255＞60で分岐せず、Aレジスタに1が代入されます。

図5—18に課題4のプログラムをモニタで実行した例をあげておきます。

8. ループを作る

さてこの章の最後に繰返しを実現させましょう。

課題5

＄６０００番地の内容と、＄６００１番地の内容をかけて、答を1バイトで＄６００２番地に格納しなさい。ただし、MUL命令は使用しないこと。

MUL命令は使用不可となっていますが、乗算は加算の繰返しで実現できます。このことを念頭に置いて作られたプログラムが図5—19です。この課題は、答の解析という形式で学習しましょう。

このプログラムは、＄６０００番地の内容の回数だけ＄６００１番地の内容を加算するという方法を取っています。

①は、まず＄６０００番地の内容、すなわち、加算の回数をBレジスタに取り込みます。これはBレジスタをカウントとして用いるからです。

②は、新しい命令です。この命令はCLR命令で、**指定したレジスタまたは指定したアドレスの内容をクリアする（0にする）命令**です。ここではAレジスタをクリアして加算に備えます。

③からがループ（繰り返し）の内側になります。ここでAレジスタに＄６００１番地の内容を加算します。つまり＄６００１番地の内容を（$6001）で表せば

A←A＋($6001)

となります（この括弧でくくるやり方で内容を表す方法はよく用いられますので覚えておいてください）。

④も新しい命令です。この命令はDEC命令の1つで、**指定したレジスタまたは指定したアドレスの内容をデクリメント（1だけ減ずる）命令**です。ここでは、Bレジスタをデクリメントしています。つまり

B←B−1

としているわけで、Bレジスタは、カウンタとして使用していたので、カウンタを1だけ減らす動作を行っています。

⑤は条件付分岐命令です。前のDEC命令では、デクリメントの結果が0になると，Zフラグがセ

〔図5−19　課題のプログラム〕

```
5000                          ORG    $5000
5000 F6    6000            ① LDB    $6000
5003 4F                    ② CLRA
5004 BB    6001    LOOP    ③ ADDA   $6001
5007 5A                    ④ DECB
5008 26    FA              ⑤ BNE    LOOP
500A B7    6002            ⑥ STA    $6002
500D 7E    AFB4            ⑦ JMP    $AFB4
                              END

0 ERROR(S) DETECTED
```

ットされます。これを利用して、カウンタであるBレジスタが0になったらループから脱出するようになっています。いい換えれば，カウンタが0でないとき＝Zフラグが0のときに分岐してループを回るということです。

⑥はAレジスタに得られた結果を＄６００２番地に格納します。

⑦は、いつものとおりのおまじないです。

理解できましたでしょうか。フローチャートにすると図5―20になります。

〔図５－22　モニタによる課題５の実行例〕

```
*D5000

5000 F6 60 00 4F BB 60 01 5A
5008 26 FA B7 60 02 7E AB F4
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■


*M6000
6000 00-03 ) 3×4
6001 00-04 )
6002 00-.

*G5000……実行

*M6000
6000 03-
6001 04-
6002 0C-■ =   12
```

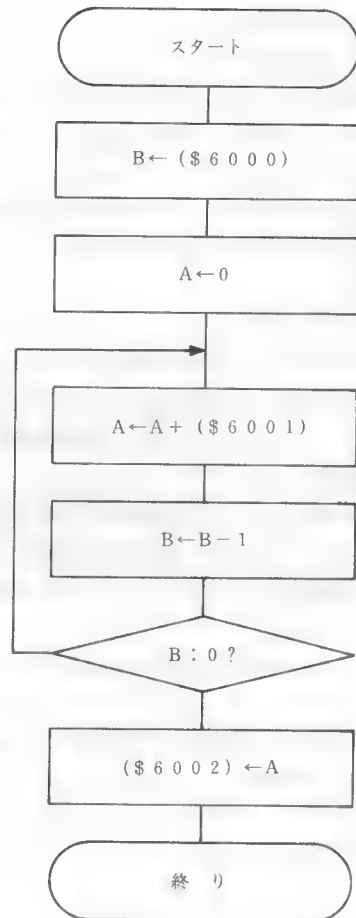



図５－20　課題５のフローチャート

〔図５－21　課題５の実行の様子〕

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDB  $6000 ┐ 初期設定
CC=00 A=00 B=03 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5003 CLRA       ┘
CC=04 A=00 B=03 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5004 ADDA $6001 ┐ ループ
CC=00 A=04 B=03 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 DECB       │  1回目
CC=00 A=04 B=02 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5008 BNE  $5004 ┘
CC=00 A=04 B=02 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5004 ADDA $6001 ┐ ループ
CC=00 A=08 B=02 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 DECB       │  2回目
CC=00 A=08 B=01 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5008 BNE  $5004 ┘
CC=00 A=08 B=01 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5004 ADDA $6001 ┐ ループ
CC=00 A=0C B=01 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5007 DECB       │  3回目
CC=04 A=0C B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5008 BNE  $5004 ┘
CC=04 A=0C B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500A STA  $6002
CC=00 A=0C B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=500D JMP  $ABF4
REF-MP TRAP AT 500D
**
            └─結果＄０Ｃ＝１２
      └─Ｚフラグがセット
```

そして、
　＄６０００番地に３
　＄６００１番地に４
を代入して実行した様子を図5—21に示します。確かに３回ループを回って結果＄０Ｃ＝12が得られているのがわかります。

モニタで同じデータを与えた実行例を図5—22に示します。いろいろな値を与えて実験してください。

さて、突然ですが、実は図5—19のプログラムには、バグ（bug：虫＝間違い）があります。少し考えてみてください。答えはあっていますが、考え方が間違っています。

バグを取ったプログラムを図5—23に示します。確かにバグが取れているかどうかは実行するなどの方法で確認してください。これは各自の独習としましょう。

このバグの解決法は数多くあります。図5—23は一つの例です。必ずしもこれがベストともいえません。参考までに本当にバグが取れているのかを試した実行例を図5—24にあげます。

〔図５－24　図５－23の実行例〕

```
*D5000

5000 4F F6 60 00 27 06 BB 60
5008 01 5A 20 F8 B7 60 02 7E
5010 AB F4 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■


*M6000
6000 00-3
6001 00-4
6002 00-.

*G5000

*M6000
6000 03-
6001 04-
6002 0C-■


*M6000
6000 03-0
6001 04-5
6002 0C-.

*G5000

*M6000
6000 00-
6001 05-
6002 00-■
```

〔図５－23　課題５のバグなし版〕

```
5000                          ORG     $5000
5000 4F                       CLRA
5001 F6    6000               LDB     $6000
5004 27    06       LOOP      BEQ     OWARI
5006 BB    6001               ADDA    $6001
5009 5A                       DECB
500A 20    F8                 BRA     LOOP
500C B7    6002     OWARI     STA     $6002
500F 7E    ABF4               JMP     $ABF4
                              END

O ERROR(S) DETECTED
```

第 6 章

マシン語プログラミング part Ⅳ

——サブルーチンとスタック——

1. JMP命令

　これまで、各プログラムの最後には暴走防止のために、おまじないとして

　JMP $ABF4

という命令を置いてきました。ここでその意味を明らかにします。

　ここで用いているのはJMP命令です。前章で分岐命令を中心に学習してきましたが、このJMP命令も一種の分岐命令ともいえます。JMP命令は、オペランドに示したアドレスにジャンプする（実行を移す）命令です。例えば、

　JMP　$500B

は、＄５００Ｂ番地にジャンプするわけです（分岐するのと同じです）。

　ここで、この命令の動作について少し述べておきましょう。まず、

　JMP　$500B

はハンドアセンブルすると、JMP命令のエクステンドモード（リラティブモードではありません）ですから、

　＄７Ｅ　＄５０　＄０Ｂ

となります。ＣＰＵはまず＄７Ｅを取り出してＪ

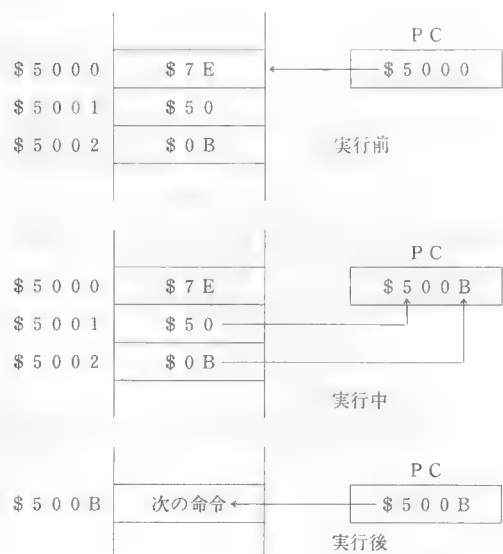



図６－１　ＪＭＰ命令の動作

MP 命令のエクステンドモードであることを認識します。次に、＄５００Ｂを取り出します。この次の動作が重要です。ＣＰＵは取り出した＄５００ＢをＰＣ（プログラムカウンタ）に代入します。すなわち

PC←$500B

とするわけです。これで命令は終了ですが、次の命令はＰＣが＄５００Ｂになっているので＄５００Ｂ番地から取り出されます。つまりジャンプしたことになります。

前章で無条件分岐命令として BRA 命令を学習しました。BRA 命令は、分岐先のアドレス指定が相対的であったのに対して、JMP 命令は、アドレスを直接指示するという違いがあります。この点から、前者を**相対分岐**、後者を**絶対分岐**と呼ぶことがあります。なぜ、両者が共に存在するのかについては、後述（第９章）することにします。

2. モニタのエントリポイント

それでは、おまじないのジャンプ先、＄ＡＢＦ４とはどういうアドレスなのでしょうか。薄々おわかりの方も多いと思いますが、この＄ＡＢＦ４というアドレスは、**モニタのエントリポイント**のアドレスなのです。

詳しく説明しましょう。その前に１章５節を再び読んでおいてください。

ＢＡＳＩＣでは、モニタに移る命令としてＭＯＮという命令を用います。ＣＰＵがＢＡＳＩＣの文をインタープリタにそって解読しＭＯＮ命令を発見したとします。すると、ＢＡＳＩＣは、ＭＯＮ命令用のサブルーチンを呼び出します。このとき呼び出すアドレスが＄ＡＢＦ４です。つまり、ＭＯＮ命令をＢＡＳＩＣで実行するということは、＄ＡＢＦ４からのマシン語を実行するのと同じになることです。

ちなみに、ＢＡＳＩＣで

ＥＸＥＣ　＆ＨＡＢＦ４⏎

とすると、モニタに制御が移り、モニタのコマンドの入力要求を示す、“＊”印が出力されます。

ですから、マシン語のプログラムの最後に、

JMP　$ABF4

を実行すると、モニタに制御が移ります。これは、今までのプログラムで

＊Ｇ５０００⏎

で＄５０００番地からのプログラムを実行の後、確かにモニタに制御が移り、“＊”が出力されることからもわかると思います。

ＢＡＳＩＣの　ＭＯＮ
＝
ＢＡＳＩＣの　ＥＸＥＣ＆ＨＡＢＦ４
＝
マシン語の　ＪＭＰ＄ＡＢＦ４

図6−3　モニタのエントリポイント

3. サブルーチン（ＪＳＲの巻）

私たちはＢＡＳＩＣのプログラムを作成する際、同じまたは同じような手順（シークエンス）が各所にあるときには、ＧＯＳＵＢ文を用いて、サブルーチンを作り、プログラムを簡略化するなどしてきました。

マシン語にもこの**サブルーチン**という概念は存

〔図６－２　モニタへのエントリ〕

```
FUJITSU F-BASIC Version 3.0
Copyright (C) 1981 By FUJITSU/MICROSOFT
30530 Bytes Free

Ready
EXEC &HABF4

*■
```

在します。

課題6

課題4の手順をサブルーチン化して、定数100、60、0に対して得られた結果を＄６００0番地から順に格納せよ。

この課題は、つまり、60との比較判定の部分をサブルーチン化して、メインルーチン部で、定数100、60、0を順に与えて、サブルーチンからの判定結果を、それぞれのアドレスに格納するということです。

まずは、そのサブルーチンの仕様を明確にしましょう。60との比較判定をサブルーチン化するのですから、仕様は次のようになります。

①入力(判定するデータ。ここでは100、60、0のこと）はAレジスタに入れてサブルーチンを呼び出す。

②出力（判定結果）は同じくAレジスタに1か0を入れて返す。

最初に、サブルーチンは完成しているとして、メインルーチンを作成することにします（このようにメインルーチンを作り、それから順に必要なサブルーチンを作っていくプログラムの作り方をトップダウンプログラミングと呼びます)。

Aレジスタに定数を与えて、サブルーチンを実行して、返された結果を指定番地に格納すればよいのですから、図6—5のようになります。

ここで用いている JSR 命令がサブルーチンを

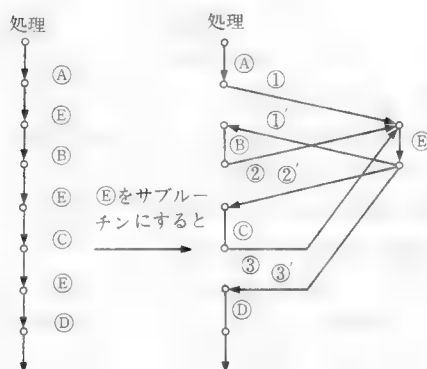



図6−4　サブルーチン化

〔図6－5　メインルーチンの作成〕

```
1     5000                    ORG   $5000  ]
2     5000 86    64           LDA   #100   ]……………… 100のときの処理
3    >5002 BD    ????         JSR   HANTEI
4     5005 B7    6000         STA   $6000  ]
5     5008 86    3C           LDA   #60    ]……………… 60のときの処理
6    >500A BD    ????         JSR   HANTEI
7     500D B7    6001         STA   $6001  ]
8     5010 86    00           LDA   #0     ]……………… 0のときの処理
9    >5012 BD    ????         JSR   HANTEI
10    5015 B7    6002         STA   $6002
11    5018 7E    ABF4         JMP   $ABF4
12
```

〔図6－6　サブルーチンの作成〕

```
13         81   3C    HANTEI  CMPA  #60
14         25   04            BLO   HIKUI  ]
15    ?    86   01            LDA   #1     ]……………… 60と比較して
16         20   02            BRA   OWARI  ]           結果をAレジスタに代入
17         86   00    HIKUI   LDA   #0     ]
18         39         OWARI   RTS   ……………………………… サブルーチンからもどる
      ↑
まだアドレスは決まっていない
```

呼び出す命令です。 JSR命令は、オペランドに示したアドレスから始まる、サブルーチンを実行する命令というわけです。オペランドには、エクステンドモードでサブルーチンのアドレスを書きます。

ここではアドレスをラベルを用いて表しています。“HANTEI”(まだ作っていない) というラベルがラベル部 (命令の左側) についているサブルーチンを、実行する予定になっています。

次に、サブルーチンを作成します。メインルーチンの方でAレジスタに判定したり、データが入っているので、まずAレジスタを60と比較します。もし60未満だったときには、Aレジスタに0を、60以上だったらAレジスタに1を代入します。ここまでは既に前章でやったことですから、すんなりと理解できるでしょう (図6—6)。

問題は次のRTS 命令です。これは、サブルーチンの実行を終って、呼び出したところへ戻れという命令です。これはBASICのRETURN文に相当します。もし、このサブルーチンが図6—4の3行めの JSR 命令から呼ばれたのであれば、4行めの命令の実行に戻ります (リターンする)。

さて、それでは両方を合わせることにします。BASICの場合と同様に (なかば習慣的に) メインルーチンの後にサブルーチンを置くことにすれば、完成したプログラムは図6—7のようになります。“HANTEI”のアドレスも決定してきちんとアセンブルされています。

4. サブルーチン (BSRの巻)

前節では、サブルーチンを呼び出すのに、JSR 命令を用いました。実はこのサブルーチンの呼び出しには JSR 命令の他に、もう1種類(2命令) あります。

前節までで無条件の分岐命令には、JMP命令と BRA命令 (LBRA 命令も含めて) がありました。違いはといえば、JMP命令がオペランドのアドレスにそのままジャンプ (分岐) するのに

JSR命令 (絶対分岐)
直接「○×△□番地」と指定する

BSR命令 (相対分岐)
相対的に「今のPC (BSR命令のあるアドレス) から, ○×バイト前(後)のアドレス」と指定する

図6−8 JSRとBSR

〔図6−7 課題6のプログラム〕

```
 1   5000                       ORG   $5000
 2   5000 86    64              LDA   #100
 3  >5002 BD    501B            JSR   HANTEI
 4   5005 B7    6000            STA   $6000
 5   5008 86    3C              LDA   #60
 6  >500A BD    501B            JSR   HANTEI
 7   500D B7    6001            STA   $6001
 8   5010 86    00              LDA   #0
 9  >5012 BD    501B            JSR   HANTEI
10   5015 B7    6002            STA   $6002
11   5018 7E    ABF4            JMP   $ABF4
12
13   501B 81    3C     HANTEI   CMPA  #60
14   501D 25    04              BLO   HIKUI
15   501F 86    01              LDA   #1
16   5021 20    02              BRA   OWARI
17   5023 86    00     HIKUI    LDA   #0
18   5025 39           OWARI    RTS
19
20                              END

 0 ERROR(S) DETECTED
```

対して(絶対分岐)、 BRA 命令は、補数表現を用いて相対的に分岐するという違いがありました(相対分岐)。

サブルーチン呼び出しにもこの絶対分岐と相対分岐とがあり、 JSR 命令は絶対分岐にあたります。そして相対分岐にあたるのが BSR 命令と LBSR 命令となります。この BSR ・ LBSR 命令は、サブルーチンのアドレス指定方法（アドレッシングモード）がリラティブモードであること以外は、 JSR 命令と違いがありません。

この BSR 命令を用いると JSR 命令の場合 3 バイトであった命令長が 2 バイトになるなどの利

〔図6－9　ＢＳＲ命令を使用した課題５〕

```
 1   5000                          ORG    $5000
 2   5000 86   64                  LDA    #100
 3   5002 8D   14                  BSR    HANTEI
 4   5004 B7   6000                STA    $6000
 5   5007 86   3C                  LDA    #60
 6   5009 8D   0D                  BSR    HANTEI
 7   500B B7   6001                STA    $6001
 8   500E 86   00                  LDA    #0
 9   5010 8D   06                  BSR    HANTEI
10   5012 B7   6002                STA    $6002
11   5015 7E   ABF4                JMP    $ABF4
12
13   5018 81   3C        HANTEI    CMPA   #60
14   501A 25   04                  BLO    HIKUI
15   501C 86   01                  LDA    #1
16   501E 20   02                  BRA    OWARI
17   5020 86   00        HIKUI     LDA    #0
18   5022 39             OWARI     RTS
19
20                                 END

 0 ERROR(S) DETECTED
```


〔図６－10　図６－９の実行の様子〕

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5000 LDA  #$64
CC=00 A=64 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5002 BSR  $5018
CC=00 A=64 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=5018 CMPA #$3C
CC=00 A=64 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=501A BCS  $5020
CC=00 A=64 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=501C LDA  #$01
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=501E BRA  $5022
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=5022 RTS
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5004 STA  $6000
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5007 LDA  #$3C
CC=00 A=3C B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5009 BSR  $5018
CC=00 A=3C B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=5018 CMPA #$3C
CC=04 A=3C B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=501A BCS  $5020
CC=04 A=3C B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=501C LDA  #$01
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=501E BRA  $5022
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=5022 RTS
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=500B STA  $6001
CC=00 A=01 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=500E LDA  #$00
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5010 BSR  $5018
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=5018 CMPA #$3C
CC=09 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=501A BCS  $5020
CC=09 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=5020 LDA  #$00
CC=05 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFD U=0000 N=01 P=5022 RTS
CC=05 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5012 STA  $6002
CC=05 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5015 JMP  $ABF4
REF-MP TRAP AT 5015
```

サブルーチン

サブルーチン

サブルーチン

点（もっと重要な利点があるのですが、その利点については後述することにします）があります。そこで前節の図6—7のプログラムの JSR 命令を BSR 命令に置き換えたものを図6—9に示します。

さて、それでは実行させてみましょう。図6—10に実行の様子、図6—11にモニタによる実行例を示します。

ここで実行の様子のうち、これまで触れなかった“N＝”の項について少し説明しておきましょう。ＢＡＳＩＣでは、サブルーチンの中からまたさらに他のサブルーチンを呼び出すということができました（これを**サブルーチンのネスト**といいます）。マシン語でもこれは可能で、サブルーチンの中でサブルーチンを呼び出してもきちんと動作します。“N＝”の項はこの**サブルーチンのネストの回数**を示しています。図6—10をみてください。初めのメインルーチンではN＝０になっています（ここではNはＣＣレジスタ内のネガティブフラグのNとは関係ありません)。すなわちサブルーチンではないのです。２行めでサブルーチンを呼び出して、サブルーチンの中に入るとN＝１となり、１重のネストであることを示しています。そして、ＲＴＳ命令でサブルーチンから戻ると、またN＝０となります。

もしN＝１のとき、さらに次の「サブルーチンのサブルーチン」を呼んだとすれば、N＝２となって２重のネストであることを示します。図6—12をみれば、その構造が理解できると思います。

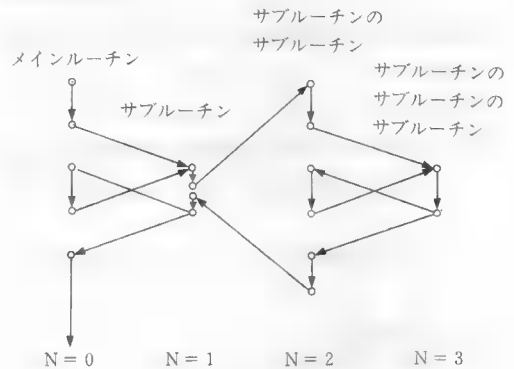



図6－12　サブルーチンのネスト

5. スタック

前節ではサブルーチンについて述べましたが、RTS 命令で、そのサブルーチンを呼び出したＪSR（または BSR、ＬＢＳＲ）命令の直後の命令に正確に戻ることができるのはなぜでしょうか。

まず最初に考えるのは、JSR （あるいは類する命令。以下も同様） 命令の際にそのときのＰＣ（プログラムカウンタ）をレジスタなどに退避する方法です。すなわち JSR 命令は、

ＴＥＭＰ←ＰＣ（次の命令のアドレス）

ＰＣ←サブルーチンのアドレス

という動作をし、 RTS 命令は

〔図６－11　モニタによる実行例〕

```
*D5000

5000 86 64 8D 14 B7 60 00 86
5008 3C 8D 0D B7 60 01 86 00
5010 8D 06 B7 60 02 7E AB F4
5018 81 3C 25 04 86 01 20 02
5020 86 00 39 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*G5000 ←……………………………………実行………図６－９のダンプリスト

*M6000
6000 01………………………１００のときの結果＝１（１００＞６０）⎫
6001 01…………………………６０のときの結果＝１（　６０＝６０）⎭ ≧６０のとき１になっている
6002 00-■………………………０のときの結果＝０（　　０＜６０）
```

PC←TEMP
という動作をすると考えるわけです。こうすれば、うまくリターンできそうです。

しかし、この方法だと、サブルーチンがネストしているときには正しく動作しなくなります（図6―13参照）。

そこで登場するのが**スタック**という概念です。まずは JSR の問題から離れて解説しましょう。このスタックというのは**マシン語で使う一時的な保存場所**といえます。構造は多少複雑かもしれませんが、非常に有用な保存場所です。

構造は、図6―14のイメージとなります。「車一台分の幅しかない袋小路型の駐車場」です。まず保存したいデータ（駐車したい車）があれば、上から入れます。このデータ（車）を①とします。さらにまた保存したいデータがあれば上から入れます（図中②）。次にデータ（車）を出すことを考えます。このときに取り出す順序は決まっています。まず②を出し、次に①を出すということです。つまり、**最後に入れたデータ（車）を最初に出す**というわけです（これをLast-In-First-Out：ＬＩＦＯといいます）。つまりスタックには取り出す順序に非常に厳しい条件があるということです。しかし、この条件があってこそ、有用な使い方ができるのも事実です。

〔図6－13　ＴＥＭＰ方式の欠点〕

```
メインルーチン
                              サブルーチン             サブルーチンの
                                                       サブルーチン
$5000  JSR  $5100 →$5100  JSR  $5200 →$5200
$5003                $5103
                         RTS                      RTS
①まず                ②次に                 ③RTS命令で
 TEMP←PC=$5003        TEMP←PC=$5103         PC←TEMP=$5103
 PC←$5100             PC←$5200
 (この時TEMP=$5003)    (この時TEMP=$5103)
                      ④RTS命令で
                       PC←TEMP=$5103！
                               だめ!!
```

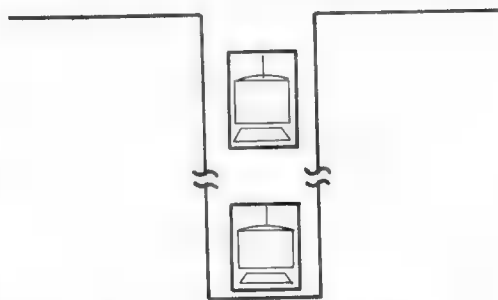



図6－14　スタックのイメージ

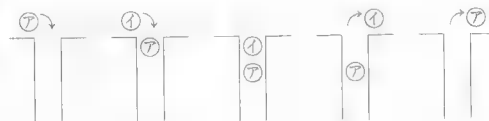
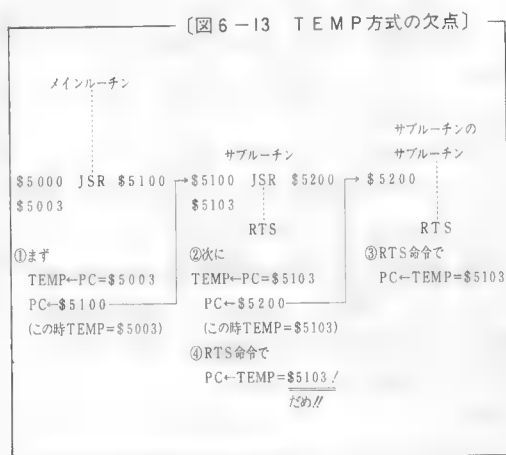



図6－15　スタックの使われ方

〔図6－16　スタックを使ったサンプル①〕

```
 1   5000               ORG   $5000
 2   5000 B6   6000     LDA   $6000
 3   5003 F6   6001     LDB   $6001
 4   5006 34   02       PSHS  A……………………Aレジスタをプッシュ
 5   5008 34   04       PSHS  B……………………Bレジスタをプッシュ
 6   500A 35   02       PULS  A……………………Aレジスタにプル
 7   500C 35   04       PULS  B……………………Bレジスタにプル
 8   500E B7   6002     STA   $6002
 9   5011 F7   6003     STB   $6003
10   5014 7E   ABF4     JMP   $ABF4
11                      END

0 ERROR(S) DETECTED
```

6. Sレジスタと Uレジスタ

それでは、少しずつスタックの構造をみていきましょう。CPUは通常の記憶装置であるメモリ上に、スタックを実現しています。この実現に大きな役割を果たしているのが、**スタックポインタ**と呼ばれるレジスタです。スタックポインタには、**Sレジスタ**と**Uレジスタ**があり1つずつスタックを実現しているので、**6809には2つのスタックがある**わけです。

5行目を終ったときの
スタックの状態

図6－17 スタック

〔図6－18　図6－16の実行例〕

```
*D5000

5000 B6 60 00 F6 60 01 34 02
5008 34 04 35 02 35 04 B7 60
5010 02 F7 60 03 7E AB F4 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■ ……………………………………………ダンプリスト


*M6000
6000 00-12 ┐
6001 00-34 │
6002 00-   ├……………………………値をセット
6003 00-.  ┘

*G5000 ……………………………………………………実行

*M6000
6000 12-
6001 34-
6002 34-  ……………………たしかに格納されている
6003 12-■
```

まずはその動作の確認をしてみましょう。図6—16は、Sレジスタのスタックを使ったプログラムの例です。新出の命令を解説しましょう。まず

　PSHS　A

は、**Sレジスタを用いたスタックにAレジスタの内容をプッシュする(スタックに積む。保存する)**命令です。PSHが命令の種類、Sがスタックポインタとして使用するレジスタ名（SレジスタかUレジスタかなので、SまたはU)、Aがスタックに内容をプッシュして保存するレジスタの名前です。ですから次の

　PSHS　B

は、Sレジスタを用いたスタックにBレジスタの内容をプッシュしています。

次に出てくる

　PULS　A

は、**Sレジスタを用いたスタックからデータをプルして(引き出す)、Aレジスタに格納する命令**です。前と同様、PULが命令の種類、Sがスタックポインタのレジスタ名、Aが格納するレジスタ名です。よって

　PULS　B

は、Sレジスタを用いたスタックからデータをプルしてBレジスタに格納する命令です。

さて、それでは図6—16のプログラムをみてみましょう。2、3行めで、＄6000番地の内容、＄6001番地の内容をそれぞれAレジスタ、Bレジスタの順でプッシュします。

次にAレジスタにプルするわけですが、LIFOの原則から、Aレジスタには、先のBレジスタの内容、すなわち＄6001番地の内容が格納されます。そして、その次の命令でBレジスタに、先のAレジスタの内容＝＄6000番地の内容が格納されています。8、9行めはもうおわかりでしょうね。

結局このプログラムは、＄6000番地の内容を＄6003番地に、＄6001番地の内容を＄6002番地に格納します。実際にモニタで実行してみた例を図6—18に示します。

7. スタックの構造

では本格的にスタックの構造にメスを入れてみましょう。まず図6—19のプログラムをみてください。このプログラムは、2行めが新しく追加されたのとPSH、PUL命令において使用するスタックポインタをSレジスタからUレジスタに変更した以外は図6—16のプログラムとかわりありません。

2行めは、これから使用するUレジスタを用いたスタックを初期化しています（後にもっと詳しく説明します)。

このプログラムを実行した様子が図6—20です。そしてモニタでの実行例が図6—21です。

まず4行めのLDB命令までの動作は問題ないでしょう。次の「 PSHU A 」は次の動作をします。

①スタックポインタであるUレジスタをデクリメント（1だけ減らす）します。

②スタックポインタであるUレジスタの内容をアドレスとするメモリに、Aレジスタの内容を格納する。

つまり、Uレジスタを＄5FFFにし（2行めで＄6000になっています)、＄5FFF番地にAレジスタの内容を格納します。さらに「 PSHS B」を実行すると同様に、図6—22のようになります。

次に「 PULS A 」は

①スタックポインタであるUレジスタの内容をアドレスとするメモリの内容を、Aレジスタに格納する。

〔図6－19　スタックを使ったサンプル②〕

```
  1    5000                            ORG    $5000
  2    5000 CE    6000                 LDU    #$6000
  3    5003 B6    6000                 LDA    $6000
  4    5006 F6    6001                 LDB    $6001
  5    5009 36    02                   PSHU   A
  6    500B 36    04                   PSHU   B
  7    500D 37    02                   PULU   A
  8    500F 37    04                   PULU   B
  9    5011 B7    6002                 STA    $6002
 10    5014 F7    6003                 STB    $6003
 11    5017 7E    ABF4                 JMP    $ABF4
 12                                    END

O ERROR(S) DETECTED
```

〔図6－20　図6－19の実行の様子〕

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFF U=0000 N=00 P=5000 LDU  #$6000
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=6FFF U=6000 N=00 P=5003 LDA  $6000
CC=00 A=12 B=00 DP=00 X=0000 Y=0000 S=6FFF U=6000 N=00 P=5006 LDB  $6001
CC=00 A=12 B=34 DP=00 X=0000 Y=0000 S=6FFF U=6000 N=00 P=5009 PSHU #$02 →PSHU A
CC=00 A=12 B=34 DP=00 X=0000 Y=0000 S=6FFF U=5FFF N=00 P=500B PSHU #$04 →PSHU B
CC=00 A=12 B=34 DP=00 X=0000 Y=0000 S=6FFF U=5FFE N=00 P=500D PULU #$02 →PULU A
CC=00 A=34 B=34 DP=00 X=0000 Y=0000 S=6FFF U=5FFF N=00 P=500F PULU #$04 →PULU B
CC=00 A=34 B=12 DP=00 X=0000 Y=0000 S=6FFF U=6000 N=00 P=5011 STA  $6002
CC=00 A=34 B=12 DP=00 X=0000 Y=0000 S=6FFF U=6000 N=00 P=5014 STB  $6003
CC=00 A=34 B=12 DP=00 X=0000 Y=0000 S=6FFF U=6000 N=00 P=5017 JMP  $ABF4
REF-MP TRAP AT 5017
**
```

＄6000番地に＄12，＄6001番地に＄34を格納してある

②スタックポインタをインクリメント（1だけ増す）する。

と動作します。つまり＄5FFF番地の内容をAレジスタに格納し，Uレジスタを＄5FFFにしています。さらに「　PULS B　」を実行した結果が図6―23に示されています。

ということは、PSH 、PUL 命令を動作させると、確かにスタックが実現できるということです。この例でもわかると思いますが、**スタックは初期化したアドレスからアドレスの小さい方（若い方）に延びていきます。**そして最初に与えたアドレス（この例では＄6000番地）の内容は使用されません。

〔図6－21　図6－19の実行例〕

```
*D5000

5000 CE 60 00 B6 60 00 F6 60
5008 01 36 02 36 04 37 02 37
5010 04 B7 60 02 F7 60 03 7E
5018 AB F4 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*D5FE0

5FE0 00 00 00 00 00 00 00 00
5FE8 00 00 00 00 00 00 00 00
5FF0 00 00 00 00 00 00 00 00
5FF8 00 00 00 00 00 00 00 00
6000 12 34 00 00 00 00 00 00
6008 00 00 00 00 00 00 00 00
6010 00 00 00 00 00 00 00 00
6018 00 00 00 00 00 00 00 00
*G5000


*D5FE0

5FE0 00 00 00 00 00 00 00 00
5FE8 00 00 00 00 00 00 00 00
5FF0 00 00 00 00 00 00 00 00
5FF8 00 00 00 00 00 00 34 12
6000 12 34 34 12 00 00 00 00
6008 00 00 00 00 00 00 00 00
6010 00 00 00 00 00 00 00 00
6018 00 00 00 00 00 00 00 00
*■
```

8. *SとUの違い*

既に述べましたが、6809には2つのスタックがあります。ここでは、それぞれの用途について解説しましょう。

まずSレジスタは本名をハードウェアスタックポインタといいSレジスタを使ったスタックは次の用途で利用されます。

図6－22　PSH命令の実行

①PSH、PUL命令を利用してレジスタの内容を一時的に保存しておく。

②JSR命令などでサブルーチンを呼んだときに、戻り番地を退避しておく。

③割込み（基礎編では解説しません）が発生したときに各レジスタを退避しておく。

このうち①は既に述べました。②が5節から懸案となっていたことですので詳しく解説しましょう。JSRなどのサブルーチンを呼び出す命令を実行すると、まずCPUはサブルーチン実行後に戻ってくるべきアドレス（図6—24でいえば＄5003）をスタック（しかもSレジスタによるスタック）にプッシュします。それからPCを変換してサブルーチンの実行に移ります。サブルーチンを終了してRTS命令の実行になると、今度はスタックからさきほどプッシュした戻りアドレスを

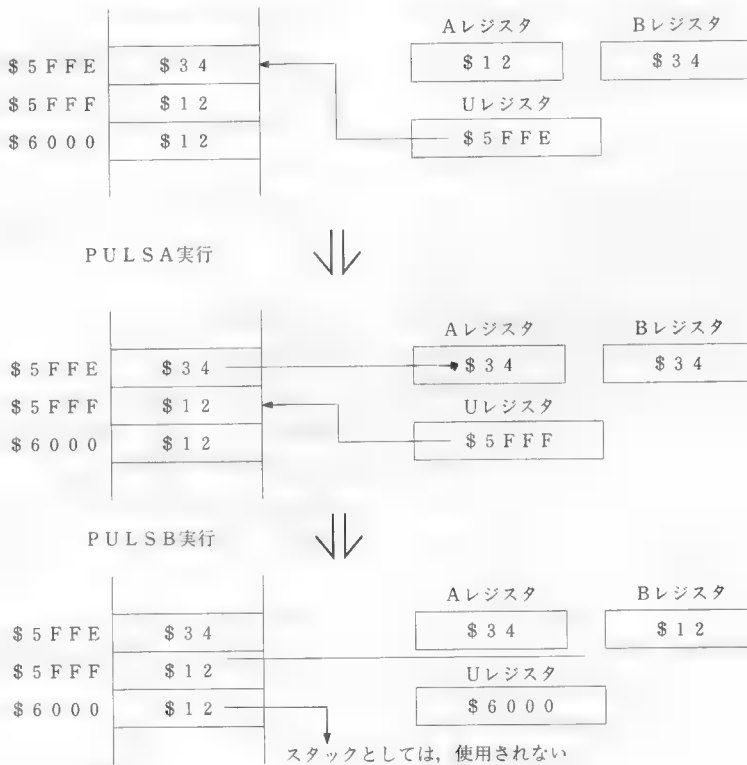



図6−23　PUL命令の実行

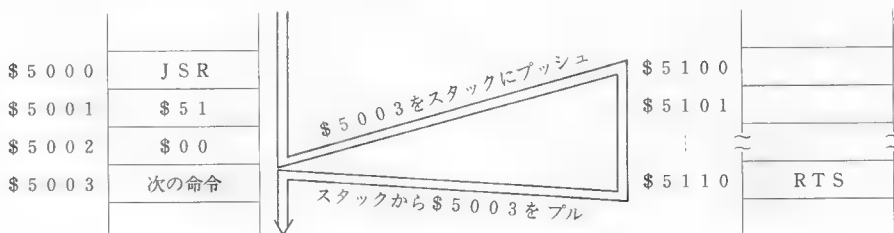



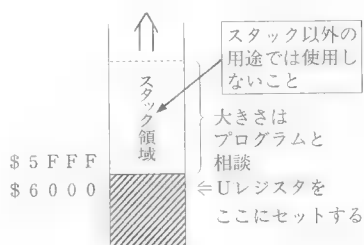
図6−24　サブルーチンとスタック

ＰＣにプルして、元のところへ戻るという仕組です。こうすると、サブルーチンがネストしていたとしてもＬＩＦＯの原則があるのでうまく処理されていきます。

③はやや難解なものなので基礎編では解説しません。

次にＵレジスタですが、これは本名を**ユーザースタックポインタ**といいます。このＵレジスタを使ったスタックは「ユーザー」の名のとおり、ユーザーである私たちが自由に使っていいスタックで、その用途も、上述の①のみに限られています。そのため、このスタックは必要ないということであれば、Ｕレジスタは単なる16ビットのレジスタとして用いても構いません。

さて、ここで図6―19のプログラムを参照してください。このプログラムではＵレジスタをスタックとして使用しています。このようにスタックとして使用する際には必ず、スタックとして使用しても構わないメモリをある程度確保しなければなりません。この例では、＄５ＦＦＦからアドレスの少い方へ向かってスタック領域として使用することにしたので、初期化に際して＄５ＦＦＦ＋１＝＄６０００をＵレジスタに格納しています。このスタック領域の大きさは自分のプログラムと相談して最大使用に＋αした大きさとする必要があります。図6―19のプログラムでは、最大でも２バイトしか使用しないので10バイトも確保すれば十分といえます（図6―25参照)。

次に図6―16のプログラムをみてください。このプログラムではＳレジスタを用いたスタックを使用しているのに、初期化がなされていません。これはＳレジスタのスタックは、前の②、③の用途に使われているので、既に初期化は終っているからです。しかし、ここで行われた初期化によって確保されたスタック領域は、その大きさがわかりません。10バイトぐらいなら追加使用しても問題はおこらないかもしれませんが、数十バイトも使用するとなるとちょっと気になります。この時には，新たに別に確保するのですが、そのときには、次の手順で行います。まず、

```
STS  $△△△△
LDS  #$××××
```

ここで、△△△△は、もとのＳレジスタの内容を保管しておくところです。そして、プログラムの終了時には、

```
LDS  $△△△△
JMP  $ABF4
```

として，元のＳレジスタの内容を復帰させます。

図6－25　スタックの初期化

9. ポストバイト

前節までのPSH 、PUL命令では、退避するレジスタが１つに限られていましたが、実は、ＰＳＨ、PUL命令にはレジスタを一度にいくつも退避する機能があります。例えば、Ａ，Ｂ，ＣＣレジスタを退避したいのであれば、

```
PSHS A, B, CC
```

というようにコンマで区切って連記することができます。また、

```
PULS A, B, CC
```

とすることで一度に復帰できます。

しかし、このレジスタの連記には制限があります。

①同じレジスタは２度連記できない。さらに、Ｄレジスタを指定した場合にはＢレジスタ、Ａレジスタを指定することはできない。

```
PSHS A, A        ……不可
PSHS D, A        ……不可
```

②連記する順序は自由で構わないが、退避される順序はあらかじめ決まっている。そのため次の

```
PSHS A, B
PSHS B, A
```

に違いは全くない。

③スタックポインタとして用いるレジスタは、自分自身を指定することはできない。

PSHS U　　……可

PSHS S　　……不可

文章で示すと複雑に聞こえるかもしれませんがハンドアセンブルしてみると、その理由がよくわかると思います。

例として

PSHS A, B, CC

を取りあげます。まず「PSHS」は付録のインストラクション表を参照すると、アドレッシングモードはイミディエイトだけです（本当はレジスタモードという他のアドレッシングモードなのですが便宜上イミディエイトモードになっています)。そしてＯＰコードは＄３４ということがわかります。

次にオペランドをアセンブルするわけですが、ここでポストバイトというものが登場します。このポストバイトでどのレジスタを退避するのかを指示するわけです。図6―26をみてください。これがポストバイトの構成です。各ビットが各レジスタに対応していて、指定するレジスタに該当するビットを１にすると、そのレジスタが指定されるというわけです。

図6－26　ＰＳＨ／ＰＵＬのポストバイト

この例ではＡ，Ｂ，ＣＣレジスタを退避するのですから、ポストバイトは

０００００１１１b＝＄０７

となります。Ｄレジスタを指定したいときは、ＡレジスタとＢレジスタを指定します。

アセンブラを使用すると、この各ビットのセット・リセットは自動的にやってくれるので、レジスタの連記の順序は自由で構わないのです。

しかし、ＣＰＵはPSH命令の場合には、ＰＣから順々に指定されたものをプッシュし、PUL命令の場合には、ＣＣレジスタから順々に指定されたものをプルします。ですからＡレジスタをプッシュしてからＢレジスタをプッシュするつもりで、

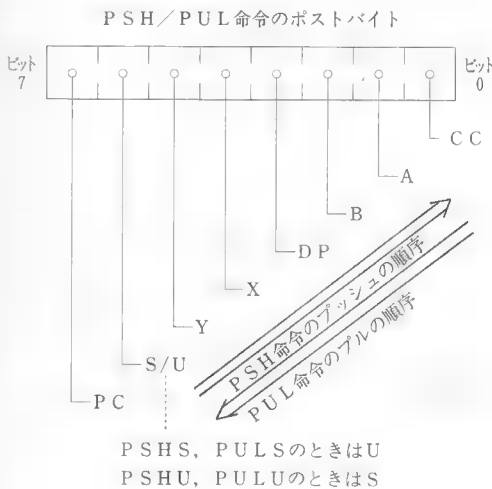
PSHS A, B

と書いてもＢ→Ａの順でプッシュされてしまいます。あくまでＡ→Ｂの順でプッシュしたいのであれば、

PSHS A

PSHS B

と２つに分けるしかありません。

第 7 章

マシン語プログラミング
partV

——アドレッシングモード——

1. アドレッシングモード

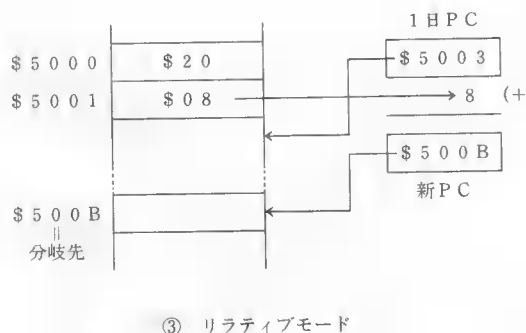
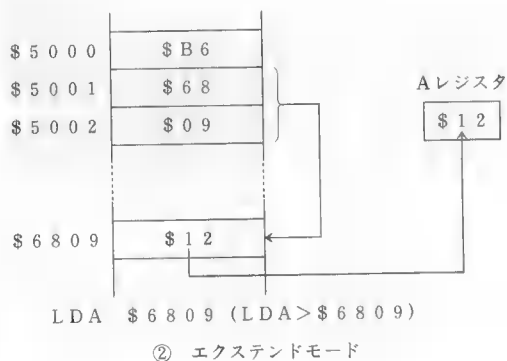
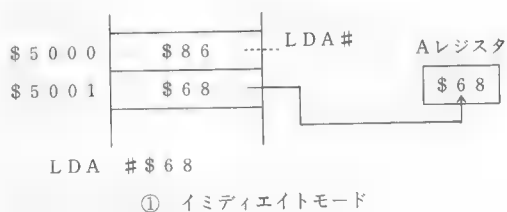
既に第3章で、アドレッシングモードとは、演算を行うときにどんな値を演算の対象にするかを示す方法であることを学びました。そこでは大まかにいって6種のアドレッシングモードがあることを示しました。そして、これまでに、イミディエイトモードとエクステンドモード、そしてリラティブモードを学習しました。ここで、この3つのアドレッシングモードを復習しておきましょう。

まずイミディエイトモードは、

LDA #$68

① イミディエイトモード

LDA $6809 (LDA>$6809)

② エクステンドモード

③ リラティブモード

BRA $500B

図7－1 既に学んだアドレッシングモード

LDA　#$68

のように用いて、オペランイドに書いた数を定数とみて演算の対象とするモードです。

次にエクステンドモードは、

LDA　$6809

のように用いて、オペランドに書いた数をアドレスとみなして、そのアドレスのメモリの内容を演算の対象とするモードです。

そして、リラティブモードは、ブランチ命令で

BRA　$500B

と用いて、オペランドに書いたアドレスに分岐します。しかし、このモードはハンドアセンブルすると形がかわり、オペランドの値をPCに加えるというモードになります。

実は、この他にもう一つだけ既に学習してしまったモードがあります。5章で出てきた

CLRA　　　　DECB

などがその例です。これらの命令は、**ニーモニックの内に演算の対象のレジスタを含んでいて、オペランドを書く必要がありません。**このようにオペランドを必要としないものを**インヘレントモード**といいます。

CLRA　を　CLR　A　と書いてはいけない。
ニーモニック　ニーモニック　オペランド

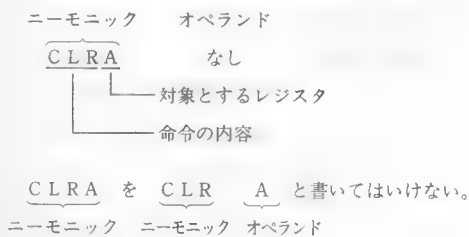



図7－2　インヘレントモード

2. ダイレクトモード

さて、いままで私たちはあるアドレスのメモリの内容を演算の対象とする場合には、エクステンドモードを用いて

LDA　$6000

のようにしてきました。エクステンドモードを用いてのメモリの参照は、(今までの例からもわかると思いますが）プログラムの中でだいぶ数多く用いられます。

今までの例では主に＄６０００番地以降のアドレスを使用してやってきたため、プログラム中に＄６０□□という記述が数多く出現しました。こうなると人間というのは無精なもので、＄６０□□のうち＄６０は省略して□□の部分だけ記入して、＄６０は暗黙の了解事項としてしまいたくなります。実は、6809ではこの省略をみとめられています。これがダイレクトモードです。それでは、詳しく説明しましょう。

ＣＰＵは上記の省略を理解してくれますが、一言も宣言しないで、突然＄６０００のつもりで＄００とやっても、動いてくれません。省略する際に、省略するのが“＄６０”であることを教えてから実行しなければいけません。その役割を果たすのが**ＤＰ（ダイレクトページ）レジスタ**です。そして、省略して下位1バイト分だけを示す（すなわち□□の部分だけを示す）アドレッシングモードを**ダイレクトモード**といいます。

例を用いて説明しましょう。

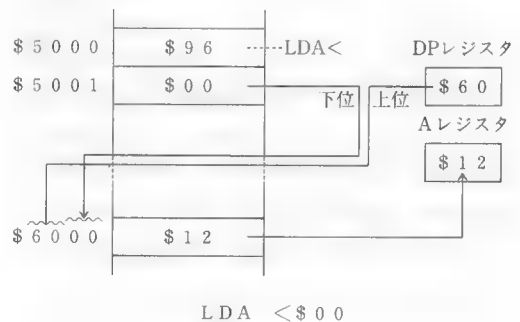



図7－3　ダイレクトモード

LDA　＄６０００
＄B６　＄６０　＄００　）アセンブル
3バイト（命令によっては4バイト）

LDA　<＄００
＄９６　＄００　）アセンブル
2バイト（命令によっては3バイト）

図7－4　ダイレクトモードの利点

LDA　<＄００

というように“<”をつけるとダイレクトモードになります。そして“<”の後には、下位の１バイト分を記述します。この命令自体にはアドレスは下位１バイトしか表記されていません。上位１バイトは、ＤＰレジスタの値が用いられます。例えばＤＰレジスタに＄６０が格納されているときに上記の命令を実行すると、＄６０００番地から値をロードしてきます（図7—3参照)。

このようにダイレクトモードを用いると、省略できて便利であることの他に、命令長も短くなり実行速度も速くなり有用です。

3. TFRとEXG

前節で述べましたが、ダイレクトモードを用いると速くて短いプログラムができるので大いに利用すべきです。しかし、これを利用するためにはＤＰレジスタに省略する値を代入しなければなりません。そのためにあるのが LDDP 命令で、ＤＰに＄６０を代入したければ、

LDDP #$60

とすればよい、といいたいところなのですが、実は LDDP 命令という命令は6809には存在しません。つまり「 LDDP　#$60　」とはいかないわけです。それで他の命令で代用するわけですが、その代用した例を図7—5のプログラムに示します。

まずこのプログラム中に出てくる新出命令につ

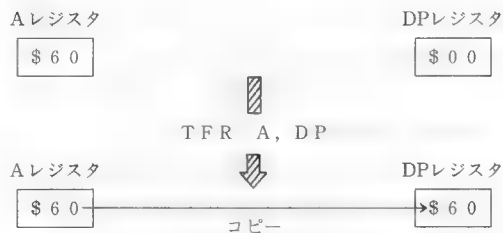



図7－6　TFR命令

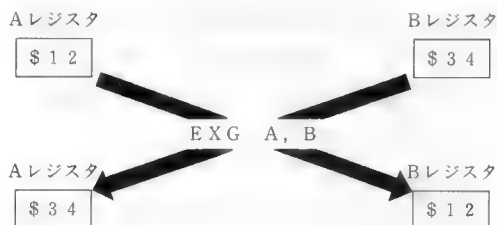



図7－7　EXG命令

TRANSFER/EXCHANGEのポストバイト

SOURCE	DESTNATION

０００＝D（A：B）	０１０１＝PC
０００１＝X	１０００＝A
００１０＝Y	１００１＝B
００１１＝U	１０１０＝CCR
０１００＝S	１０１１＝DPR

図７－８

〔図7—5　ＤＰレジスタの設定〕

```
 1   5000                ORG   $5000
 2   5000 34   0E        PSHS  D,DP   ……レジスタ退避
 3   5002 86   60        LDA   #$60 }
 4   5004 1F   8B        TFR   A,DP }  ……ＤＰレジスタの設定
 5   5006 96   00        LDA   <$00
 6   5008 D6   01        LDB   <$01
 7   500A 1E   89        EXG   A,B
 8   500C 97   00        STA   <$00
 9   500E D7   01        STB   <$01
10   5010 35   0E        PULS  D,DP   ……レジスタ復帰
11   5012 7E   ABF4      JMP   $ABF4  ……モニタへ
12                       END

0 ERROR(S) DETECTED
```

いて解説しておきましょう。

TFR 命令は、オペランドに書いてある2つのレジスタ間で転送を行う命令です。転送の方向は左から右ですから、プログラム中の

TFR　A, DP

はAレジスタの内容をDPレジスタに転送（代入、コピー）します。

EXG 命令は、オペランドに書いてある2つのレジスタの内容を交換する命令です。ですから、プログラム中の

EXG　A, B

はAレジスタの内容とBレジスタの内容を交換するわけです。

ここで出た2つの命令は、オペランドにレジスタ名を記述する命令でした。この2つとすでに述べたPSH、PUL命令は、レジスタモードと呼ばれるのですが、便宜上イミディエイトモードに分類することにします（場合によってインヘレントモードに分類することもあります)。ですからハンドアセンブルする際にはイミディエイトの項をみます。例として「 TFR A, DP 」をハンドアセンブルしてみましょう。まずインストラクション表(付録)でTFRのOPコードをみると＄1Fであることがわかります。次にオペランドですが、ここでもポストバイトが登場します。TFR／EXGのポストバイトは図7―8のようになっています。ですからこの場合は、

10001011b＝＄8B

となります。ここで注意しなければならないのは長さの違うレジスタに対してTFR／EXG命令は使用できないということです。ですから

EXG　S, A　＝＄1E、＄48

ということはできないわけです。

ちなみに、TFR命令では転送の方向は決まっていますが、EXG命令では関係ないので

EXG　A, B　＝＄1E　＄89

EXG　B, A　＝＄1E　＄98

は両方とも同じ結果をもたらします。

図7－10　図7－5の実行例

```
*d5000

5000 34 0E 86 60 1F 8B 96 00
5008 D6 01 1E 89 97 00 D7 01
5010 35 0E 7E AB F4 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
```

ダンプリスト

```
*M6000
6000 00-12 }
6001 00-34 } ······················ 定数をセット
6002 00-.

*G5000 ································ 実行

*M6000
6000 34-   } ······················ 確かに交換されている
6001 12-■  }
```

図7－9　図7－5の実行の様子

```
                                                                                         対象とした
                                                                                         アドレス
CC=00 A=AA B=BB DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 PSHS #$0E
CC=00 A=AA B=BB DP=00 X=0000 Y=0000 S=C07C U=0000 N=00 P=5002 LDA  #$60
CC=00 A=60 B=BB DP=00 X=0000 Y=0000 S=C07C U=0000 N=00 P=5004 TFR  A,DP
CC=00 A=60 B=BB DP=60 X=0000 Y=0000 S=C07C U=0000 N=00 P=5006 LDA  $00 ($6000)
CC=00 A=12 B=BB DP=60 X=0000 Y=0000 S=C07C U=0000 N=00 P=5008 LDB  $01 ($6001)
CC=00 A=12 B=34 DP=60 X=0000 Y=0000 S=C07C U=0000 N=00 P=500A EXG  A,B
CC=00 A=34 B=12 DP=60 X=0000 Y=0000 S=C07C U=0000 N=00 P=500C STA  $00 ($6000)
CC=00 A=34 B=12 DP=60 X=0000 Y=0000 S=C07C U=0000 N=00 P=500E STB  $01 ($6001)
CC=00 A=34 B=12 DP=60 X=0000 Y=0000 S=C07C U=0000 N=00 P=5010 PULS #$0E
CC=00 A=AA B=BB DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5012 JMP  $ABF4
REF-MP TRAP AT 5012
**
```

退避してあったので
実行前と同じ

ここまでくれば図7—5のプログラムの意味もわかります。まずこのプログラム中で使用(＝破壊)するレジスタをスタックに退避します。そして次の２行でＤＰレジスタに＄６０をセットします。５、６、８、９行めで、ダイレクトモードを用いています。10行めでは退避したレジスタを元に戻し（復帰）て、モニタへ飛びます。結局このプログラムはＣＣレジスタ以外のレジスタを破壊せずに＄６０００番地の内容と＄６００１番地の内容を交換するプログラムです。

図7—9に実行の様子、図7—10にモニタによる実行例を示します。

4. インデックスモード

大別して６種あるアドレッシングモードのうち、既に５種を学習したので，残るはあと１種のみです。しかしこの一種がくせもので細かく分けると９種類、さらに細かく分類すると24種にもなります。ここからは、その残る一種を少しずつ解説していくことになります。

さて、前々節でダイレクトモードを取りあげました。これは上位８ビットをＤＰレジスタで決定し，下位８ビットをユーザー（私たち）が指定してアドレスを得る方法でした。これを用いると実行速度もあがり良い方法でなのですが、性質上、＄ＸＸ００～＄ＸＸＦＦまでをダイレクトモードで使用することはできても、例えば＄６０２０～＄６１１Ｆまでをダイレクトモードで使用するということはできず、ダイレクトモードで使用するアドレスには制限があります。また、データ（例えば表など）が256バイトを超えると、256バイトを越える場合にはＤＰレジスタを再設定するか、エクステンドモードを用いるかしなければなりません。この用な制限からダイレクトモードは主に、一時記憶用などの準レジスタ的な用途に用いられます。

では、表やＢＡＳＩＣの配列みたいなものを取りあつかうのに、なにか便利な方法はないかというわけで登場するのが、インデックスモードです。インデックス（Index）が、指標と訳されるように**このインデックスモードはアドレスに指標をつけ、その指標から前にいくつ、後にいくつというようにアドレスを指定するという方法です。**

5. Xレジスタ，Yレジスタ

前節で「指標」と述べました。この指標（見出し、印、etc）となるのが **インデックスレジスタ** という16ビットのレジスタです。このインデックスレジスタはＤＰレジスタとは違って16ビットなので、＄００００～＄ＦＦＦＦのＣＰＵのアドレスのどこにでもこの指標をつけることができます。

6809の先祖である6800では、インデックスレジスタは１つしかなかったのですが、6809では、Ｘレジスタ（正式にはインデックスレジスタＸ）とＹレジスタ（同様）の２つがあり、両者には機能的な差異はなく同等に扱えます。

図７－11　インデックスモード

```
    1    5000                         ORG    $5000
    2    5000 8E    6000              LDX    #$6000
    3    5003 A6    84                LDA    ,X
    4    5005 81    3C                CMPA   #60
    5    5007 25    04                BLO    HIKUI
    6    5009 86    01                LDA    #$01
    7    500B 20    01                BRA    OWARI
    8    500D 4F           HIKUI      CLRA
    9    500E A7    01     OWARI      STA    1,X
   10    5010 7E    ABF4              JMP    $ABF4
   11                                 END

 0 ERROR(S) DETECTED
```

さらに6809で特徴的であるのは、**スタックポインタであったSレジスタとUレジスタも**（インデックスレジスタではないけれども）**指標として使用できます**。特にUレジスタは、スタックポインタとして用いないのであれば、XレジスタやYレジスタと（一部の命令のそのまた一部を除いて）同等に扱えます。どうしてスタックポインタをインデックスレジスタとして使用できるのが特徴的であるのかと思われるかもしれませんが、これがあると非常に役に立つ場面があるのです。

まず図7—11をみてください。このプログラムは、図5－14のプログラムを、インデックスモードを用いて書き換えたものです。

2行めにいきなり

LDX　#$6000

という命令が出現します。すでにＬＤ命令は説明済なので、この命令がXレジスタに＄６０００を代入する命令であることはわかると思います。ここでは指標として用いるXレジスタに＄６０００を代入することによって、＄６０００番地に、指標をつけて、この＄６０００番地付近のアドレスは、指標を基準にして、参照するようにしようというわけです。

6. ゼロオフセット

図7—11のプログラムの2行めで、＄６０００番地に指標をつけたわけですが、次にその指標を基準にしてメモリを読み書きする方法を説明します。

図7—11の3行めをみてください。オペランドに見慣れない表記があります。このオペランドの表記は、Xレジスタの内容をアドレスとするメモリを演算の対象にするということを示しています。ですから、このプログラムのこの命令では、＄６０００番地の内容をAレジスタにロードするということになります（図7—12参照）。

このように、**インデックス用レジスタ（X、Y、U、Sの各レジスタのことを示すことにします。以下同様）の内容をアドレスとするメモリを演算の対象にする**——すなわち、指標(インデックス)の指すメモリを演算の対象にするアドレスの指定方法を、**ゼロオフセット**といいます。このゼロオフセットの名前の意味については次節に譲ることにしましょう。

この例では**X**レジスタを指標としましたが、他のインデックス用レジスタも用いることができます。この場合，**オペランドには、“，”（カンマ）の後に指標とするレジスタ名を書きます**。例えば、AレジスタにXレジスタの示すメモリの内容を加えるには

ADDA ,X

とすればよいわけです。もし、対象となるデータが16ビットの場合、例えば

STD　,Y

などのときには、Yレジスタの指すメモリと、その次のアドレスのメモリが参照されます。これはエクステンドモードの場合と同じです。

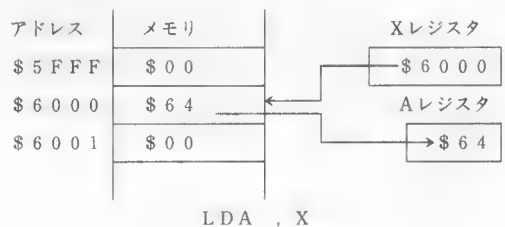



図7－12　ゼロオフセット

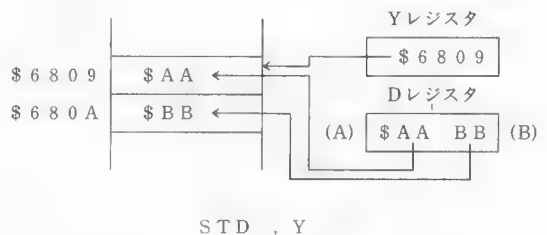



図7－13　ゼロオフセット(16ビットの場合)

7. 定数オフセット

図7—11のプログラムに戻りましょう。4行めから8行めまでは、すでに第5章で述べたものと全く同じですから、説明しません。もし記憶があやふやでしたら5章を復習してください。

問題は9行めです。前節のゼロオフセットと似

ていますが、少し違い、**定数オフセット**と呼ばれるものです。

これは、指標を中心として前後のアドレスを読み書きする方法です。この例、すなわち

STA 1,X

では、Xレジスタの値に1を加えた値をアドレスとするメモリにAレジスタの値をストアするということになっています。

このように、**インデックス用レジスタの内容と定数を加えた値をアドレスとするメモリを演算の対象とする**、すなわち(指標とするレジスタの値＋定数)の指すメモリを演算の対象とするアドレスの指定方法を定数オフセットといいます。

ここで出てきた定数とは、－32768～＋32767の数(16ビット符号つき)とすることができるので、例えば、＄６０００に指標がついているときに＄５ＦＦＦ番地を利用したいのであれば、定数として－１をとればよいわけです。この定数オフセットの表記法としては、オペランドにまず定数を書き、“，”(コンマ)で区切って指標とするレジスタ名を書きます。例えば

ADDA －1,X

というようにすればよいわけです。

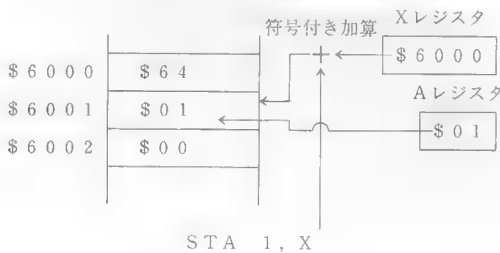



図7－14 オフセット

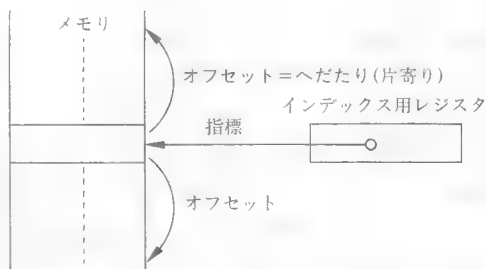



図7－15 オフセットとは

前節で説明したゼロオフセットも定数オフセットの一種で、定数が０の定数オフセットとみなすことができます。しかし、

LDA 0,X

と

LDA ,X

では、若干の違いがあります。これについては次節で、明らかになるでしょう。

8. ゼロオフセットのハンドアセンブル

定数オフセットを学習したところで、インデックスアドレッシングのハンドアセンブルの仕方をみてみます。6809は、ブランチ命令などハンドアセンブルだとわずらわしい命令を数多く持っており、ある意味で「ハンドアセンブル向きでないＣＰＵ」といえます。しかし、マシン語を学習する際には、どのようにアセンブルされるのかを知るのは重要ですから、避けて通ることはできません。

まず

LDA ,X

を例に取ってやってみましょう。この命令のＯＰコードは，LDA命令のインデックスモードであることから、インストラクション表より、＄Ａ６であることがわかります。次にオペランド部分ですが、このインデックスモードにおいては、ＴＦＲ命令などで使用した**ポストバイト**と呼ばれるＯＰコードの拡張部分を使用します。図7－16がインデックスモードのポストバイトです。まだ学習していないモードもありますが、この場合は、ゼロオフセットですので、**コンスタント(定数)オフセットのオフセットなし＝ゼロオフセット**の項をみてください。「インダイレクトでない場合」と「インダイレクトの場合」とがありますが、これは「インダイレクトでない場合」です(詳しくは後述します)。アセンブラ形式の項をみると「 ,R 」となっており、「 ,X 」と一致します(Ｒはインデックス用レジスタを示しています)。

そこでポストバイトは、この表から

$1\,r_1\,r_2\,0\,0\,1\,0\,0\,b$

となります。ここで「$r_1\,r_2$」は指標とするレジス

タによって定まります。この例ではXレジスタを用いているので（図7—16下の記号の項参照）「00」です。よって、求めるポストバイトは

　10000100b＝$84

となり、結局全体で

　$A6　$84

となるわけです。

9. 定数オフセットのハンドアセンブル

次に

　STA　1,X

をハンドアセンブルしてみます。図7—16を見ると定数（コンスタント）オフセットの項には4種の定数オフセットがあるのがわかります。このうち「オフセットなし」はゼロオフセットのことですから、この場合（オフセット＝1）のときには利用できません。すると、残りは3種ということになりますが、この場合だとこの3種全てを用いることができます。まずOPコードは STA 命令のインデックスモードですから、$A7となります。次にオペランドですが、これは1種類ずつ調べてみましょう。

◀**5ビットオフセット**▶

ポストバイトは、図7—16から

　$0r_1r_2nnnnnb$

であることがわかります。「r_1r_2」はXレジスタですから「00」です。問題は「nnnnn」の部分です。ここには、オフセットの値（ここでは1）を**5ビットの補数表現**で示します。

　1→00001

ですので、ポストバイトは、結局、

アドレッシング モード			インダイレクトでない場合				インダイレクトの場合			
			アセンブラ形式	ポスト バイト	+~	+#	アセンブラ形式	ポスト バイト	+~	+#
インデックスト	コンスタントオフセット	オフセットなし	,R	1RR00100	0	0	〔,R〕	1RR10100	3	0
		5ビット オフセット	n, R	0RRnnnnn	1	0	, —	—	—	—
		8ビット オフセット	n, R	1RR01000	1	1	〔n, R〕	1RR11000	4	1
		16ビット オフセット	n, R	1RR01001	4	2	〔n, R〕	1RR11001	7	2
	アキュムレータオフセット	Aレジスタ オフセット	A, R	1RR00110	1	0	〔A, R〕	1RR10110	4	0
		Bレジスタ オフセット	B, R	1RR00101	1	0	〔B, R〕	1RR10101	4	0
		Dレジスタ オフセット	D R	1RR01011	4	0	〔D, R〕	1RR11011	7	0
	オートインクリメント／デクリメント	インクリメント(+1)	,R+	1RR00000	2	0	—	—	—	—
		インクリメント(+2)	,R++	1RR00001	3	0	〔,R++〕	1RR10001	6	0
		デクリメント (−1)	,−R	1RR00010	2	0	—	—	—	—
		デクリメント (−2)	,−−R	1RR00011	3	0	〔,−−R〕	1RR10011	6	0
プログラムカウンタ リラティブ		8ビット オフセット	n, PCR	1XX01100	1	1	〔n, PCR〕	1XX11100	4	1
		16ビット オフセット	n, PCR	1XX01101	5	2	〔n, PCR〕	1XX11101	8	2
エクステンディット インダイレクト		16ビット アドレス	—	—	—	—	〔n〕	10011111	5	2

〈記号〉　R：X　RR：00＝X　X：Don't care　±＝追加されるマシンサイクル数

　　　　　Y　　　01＝Y　　　　　　　　　+#＝追加されるバイト数

　　　　　U　　　10＝U

　　　　　S　　　11＝S

図7−16　インデックスアドレッシングモードのポストバイト

0 0 0 0 0 0 0 1 b＝$01

全体で

$A 7　$01

となります。もう1つ例を図7—17①に示します。

◀8ビットオフセット▶

ポストバイトは、図7—16から

1 r_1 r_2 0 1 0 0 0 b

となり、Xレジスタですから「$r_1 r_2$＝0 0」となります。ここで図7—16の「＋#」の項をみてください。この項は、**インストラクション表の「#」の項にさらに加える値を示しています。**「#」は命令長ですが、STA命令のインデックスモードの項には「2＋」と書かれています。これに、8ビットオフセットの「＋#」の項の「1」を加えると全体での命令長は

2＋1＝3バイト

となります。

さて、OPコードとポストバイトは、

$A 7　$8 8

で、オフセット値は8ビットの補数表現で示すので、全体では

$A 7　$8 8　$0 1

となります。もう1つの例を図7—17②に示します。

◀16ビットオフセット▶

ポストバイトは、図7—16から

1 r_1 r_2 0 1 0 0 1 b

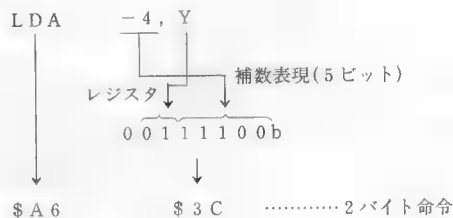
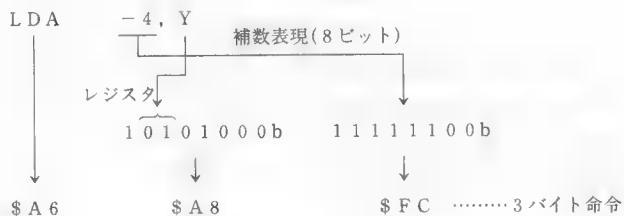
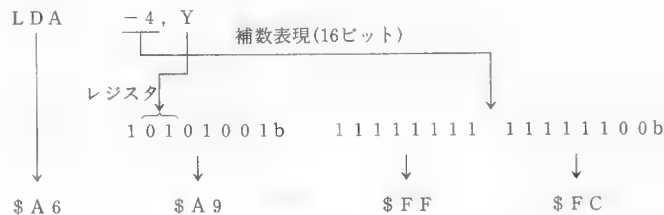
となり、Xレジスタであることから「$r_1 r_2$＝0 0」

① 5ビットオフセット

② 8ビットオフセット

③ 16ビットオフセット

図7－17　定数オフセット

となります。オフセットは今度は16ビットの補数表現で示しますので、オフセットだけで2バイト必要とするので、全体で「2＋2バイト」の命令長となり、この例では

　$A7　$89　$00　$01

となります、もう1つ例を図7—17③に示します。

　これまで述べてきたように、

　STA　1,X

では、3種類のアセンブルが可能ですが、当然のことですが、全て正常に動作します。

10. 定数オフセットの選択

　前節で

　STA　I,X

を3種にハンドアセンブルしましたが、どれを選択すべきでしょうか。

　既にみてきたように命令長は、5ビットオフセットなら2バイト、8ビットオフセットなら3バイト、16ビットオフセットならば4バイトとなっています（命令によってはそれぞれ1バイト長くなることがあります。例えばLDY命令)。命令長はメモリを占める長さですから、短い程よいということになります。また命令長は、命令の実行速度にも影響しますから、加えて短いことが望まれます。ですから前節の例の場合には5ビットオフセットが最適というわけです。

　しかし、オフセット値の大きさによって、5ビットオフセットや8ビットオフセットでは表現できない場合もあります。ですから、どの定数オフセットを用いるかは、オフセット値と相談して決定する必要があるわけです。一般に図7—18で示したように選択します。定数オフセットを用いる場合には、どの定数オフセットを用いるのかを選択しなければなりません。しかし、この選択を人間が行わなければならないのはハンドアセンブルの

オフセット値	用いるオフセット
0	ゼロオフセット
−16 ～ ＋15	5ビットオフセット
−128 ～ ＋127	8ビットオフセット
その他	16ビットオフセット

図7－18　定数オフセットの選択

図7－20　図7－11の実行例

```
*D5000

5000 8E 60 00 A6 84 81 3C 25
5008 04 86 01 20 01 4F A7 01
5010 7E AB F4 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■


*M6000……………………………………１００をセット
6000 00-64
6001 00-.

*G5000……………………………………………実行

*M6000
6000 64-
6001 01-■……………確かに１００＞６０なので，
                      ＄０１となっている
```

図7－19　図7－11の実行の様子

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDX  #$6000
CC=00 A=00 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5003 LDA  ,X
CC=00 A=64 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 CMPA #$3C
CC=00 A=64 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 BCS  $500D
CC=00 A=64 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5009 LDA  #$01
CC=00 A=01 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500B BRA  $500E
CC=00 A=01 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500E STA  $1,X
CC=00 A=01 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5010 JMP  $ABF4
REF-MP TRAP AT 5010
**
```

際のみで、アセンブラを使用する場合には、通常アセンブラの方で最適な定数オフセットを選択してくれるので、人間は気にすることはありません。

さて、完成した図7—11のプログラムを実行した様子とモニタでの実行例を図7—19, 7—20にあげます。

11. アキュームレータオフセット

さて、ここまではオフセットが定数である場合、すなわち静的な場合について扱ってきました。しかし、静的にしかオフセットを扱えないと、あるアドレスから連続的に読み書きしたい場合などは不便です。そこで動的なオフセットを扱えるように6809では、アキュームレータオフセットと呼ばれるモードを持っています。

アキュームレータオフセットには、図7—16にありますが、Aレジスタオフセット、Bレジスタオフセット、Dレジスタオフセットの3種類があり、アセンブラ表記は、それぞれ「A,R」「B,R」「D,R」となっています。

このモードは、定数オフセットでは定数で与えていたオフセットを、各アキュームレータで与えるモードです。例をあげると、

LDA　B,X

でBレジスタの値が－1＝＄FFならば、動作的には

LDA　−1,X

と同じというわけです。すなわち、インデックス用レジスタの値に、指定したアキュームレータの内容を補数表現で加算した値をアドレスとするメモリに対して演算を施すというわけです。

ここで注意してほしいのはアキュームレータの値は補数表現とみなされるということです。例えばXレジスタが＄６０００のときに＄６０ＦＦ番地をロードするつもりで、Bレジスタに＄ＦＦを代入して

LDA　B,X

としてもだめで、この場合には＄５ＦＦＦ番地をロードしてしまいます。

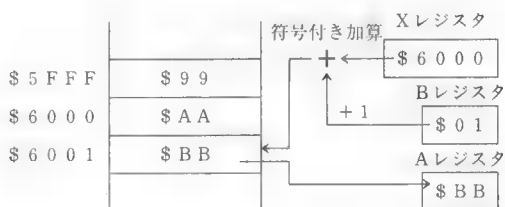
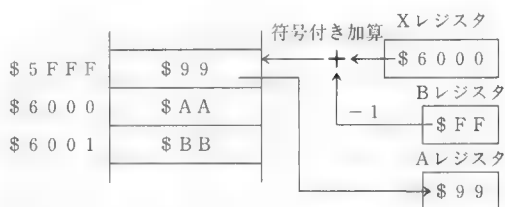
では実際の例でみてみましょう。図7—22にサン

ＬＤＡ　Ｂ，Ｘ（Ｂレジスタ＝＄０１＝＋１）

ＬＤＡ　Ｂ，Ｘ（Ｂレジスタ＝＄ＦＦ＝－１）

図7−21　アキュームレータオフセット

図７−22　アキュームレータ・オフセットの例

```
  1   5000                       ORG    $5000
  2   5000 8E   6000             LDX    #$6000
  3   5003 5F                    CLRB
  4   5004 4F                    CLRA
  5   5005 AB   85       LOOP    ADDA   B,X
  6   5007 5C                    INCB
  7   5008 C1   0A               CMPB   #10
  8   500A 26   F9               BNE    LOOP
  9   500C A7   85               STA    B,X
 10   500E 7E   ABF4             JMP    $ABF4
 11                              END

0 ERROR(S) DETECTED
```

プルプログラムを示します。このプログラムは、＄６０００番地から始まる10バイトのメモリ(＄６０００〜＄６００９番地　の内容を合計して、11バイトめ（＄６００Ａ番地）に格納するプログラムです。

まずXレジスタによって＄６０００番地に指標をつけて、カウンタとして用いるBレジスタと、合計を保持するAレジスタをクリアします。

次が核心の部分です。この命令で（X＋B）番地の内容をAレジスタに加えます。次にBレジスタをインクリメント（１だけ増します。新しく出た命令ですが、動作は

Bレジスタ←Bレジスタ＋1

となります。　DEC Bの逆です）して、Bレジスタが10でなければ、ループします。Bレジスタが10ならばループは終了ですので、ループを抜けて結果を（X＋＄A）番地に格納します。

このプログラムを実行した様子を図7―23と図7―24に示します。図7―24で確かに10回ループしているのがよくわかると思います。

図7－23　図7－22の実行例

```
MON

*D5000

5000 8E 60 00 5F 4F AB 85 5C
5008 C1 0A 26 F9 A7 85 7E AB
5010 F4 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■



*D6000          1＋2＋……＋9＋10を実行させる

6000 01 02 03 04 05 06 07 08
6008 09 0A 00 00 00 00 00 00
6010 00 00 00 00 00 00 00 00
6018 00 00 00 00 00 00 00 00
6020 00 00 00 00 00 00 00 00
6028 00 00 00 00 00 00 00 00
6030 00 00 00 00 00 00 00 00
6038 00 00 00 00 00 00 00 00
*G5000………………………………………………→実行

*D6000
                 結果＄37＝55が格納されている.
6000 01 02 03 04 05 06 07 08
6008 09 0A 37 00 00 00 00 00
6010 00 00 00 00 00 00 00 00
6018 00 00 00 00 00 00 00 00
6020 00 00 00 00 00 00 00 00
6028 00 00 00 00 00 00 00 00
6030 00 00 00 00 00 00 00 00
6038 00 00 00 00 00 00 00 00
*■
```

12. オートインクリメント，デクリメント

アキュームレータオフセットでは、オフセット値を動かすことによって動的なものを実現しまし

図7－24　図7－22の実行の様子

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDX  #$6000
CC=00 A=00 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5003 CLRB
CC=04 A=00 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5004 CLRA
CC=04 A=00 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X    ┐1回目
CC=00 A=01 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB        │＄6000
CC=00 A=01 B=01 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A   │番地を加算
CC=09 A=01 B=01 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005  ┘
CC=09 A=01 B=01 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X    ┐2回目
CC=00 A=03 B=01 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB        │＄6001
CC=00 A=03 B=02 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A   │番地を加算
CC=09 A=03 B=02 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005  ┘
CC=09 A=03 B=02 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X    ┐3回目
CC=00 A=06 B=02 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB        │＄6002
CC=00 A=06 B=03 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A   │番地を加算
CC=09 A=06 B=03 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005  ┘
CC=09 A=06 B=03 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X    ┐4回目
CC=00 A=0A B=03 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB        │＄6003
CC=00 A=0A B=04 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A   │番地を加算
CC=09 A=0A B=04 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005  ┘
```

```
CC=09 A=0A B=04 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X     ┐5回目
CC=00 A=0F B=04 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB         │$6004
CC=00 A=0F B=05 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A    │番地を加算
CC=09 A=0F B=05 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005   ┘
CC=09 A=0F B=05 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X     ┐6回目
CC=20 A=15 B=05 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB         │$6005
CC=20 A=15 B=06 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A    │番地を加算
CC=29 A=15 B=06 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005   ┘
CC=29 A=15 B=06 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X     ┐7回目
CC=00 A=1C B=06 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB         │$6006
CC=00 A=1C B=07 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A    │番地を加算
CC=09 A=1C B=07 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005   ┘
CC=09 A=1C B=07 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X     ┐8回目
CC=20 A=24 B=07 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB         │$6007
CC=20 A=24 B=08 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A    │番地を加算
CC=29 A=24 B=08 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005   ┘
CC=29 A=24 B=08 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X     ┐9回目
CC=00 A=2D B=08 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB         │$6008
CC=00 A=2D B=09 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A    │番地を加算
CC=09 A=2D B=09 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005   ┘
CC=09 A=2D B=09 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 ADDA B,X     ┐10回目
CC=20 A=37 B=09 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5007 INCB         │$6009
CC=20 A=37 B=0A DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5008 CMPB #$0A    │番地を加算
CC=24 A=37 B=0A DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500A BNE  $5005   ┘
CC=24 A=37 B=0A DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500C STA  B,X      $600A
CC=20 A=37 B=0A DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=500E JMP  $ABF4    番地に結果
REF-MP TRAP AT 500E                                                         を格納
**
```

たが、オートインクリメントとオートデクリメントは、指標の方を動かすことによって動的なものを実現しようというものです。

図7—16にあるとおり、**オートインクリメント**と**オートデクリメント**に各2種類のモードがあります。

まずオートインクリメントについて解説します。これは途中まではゼロオフセットの動作と同じで指標としたレジスタの示すアドレスのメモリに演算を施します。その後で、指標としたレジスタをインクリメントします（これを**ポストインクリメント**といいます)。例をあげてみましょう。

LDA　,X＋

で実行前にXレジスタが＄６０００だったとすると、Aレジスタに＄６０００番地の内容をロードし、その後、Xレジスタをインクリメントします。

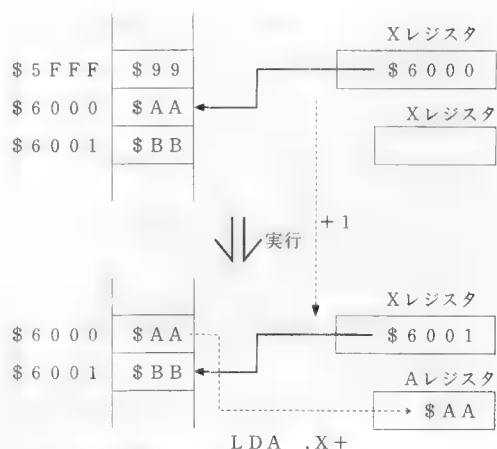



図7－25　オートインクリメント

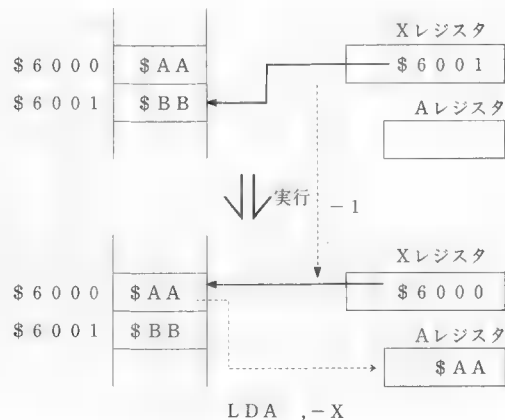



図7－26　オートデクリメント

実行後、Xレジスタは＄６００１になります。

もし、「 ,X＋ 」でなく「 ,X＋＋ 」であれば演算後にXレジスタは＋２されます。

次にオートデクリメントの場合ですが、この場合には演算の前にデクリメントしてから（プレデクリメント）演算を施します。ですから＄６００１がXレジスタにセットされている状態で

LDA ,−X

を実行すると、まずXレジスタがデクリメントされて＄６０００になり、その後に演算を行い、＄６０００番地の内容をロードします。

この場合も「 ,−X 」でなく「 ,−−X 」であれば、Xレジスタを−２したあと演算を施します。

注意しなければならないのは、オートインクリメントの場合には演算後、オートデクリメントの場合には演算前にインデックス用レジスタが変化するということです。実はこれは、既にみたスタックの動作と同じ形式になっているのです。アセンブラ表記では、「＋」ならインデックス用レジスタ名の後、「−」なら前に置かれるので区別しやすくなっています。

図7—27にこのモードを使ったサンプルプログラムを示します。このプログラムは前節の図7—22のプログラムと同じ結果をもたらすプログラムです。実行例を図7—28と図7—29に示しますので研究してください。

13. インダイレクト

再び、図7—16に戻ります。ここまではこの図の

〔図７−28　図７−27の実行例〕

```
MON

*D5000

5000 8E 60 00 C6 0A 4F AB 80
5008 5A 26 FB A7 84 7E AB F4
5010 00 00 00 00 00 00 00 00
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*■
        ダンプリスト

*D6000
        1＋2＋…＋9＋10を実行
6000 01 02 03 04 05 06 07 08
6008 09 0A 00 00 00 00 00 00
6010 00 00 00 00 00 00 00 00
6018 00 00 00 00 00 00 00 00
6020 00 00 00 00 00 00 00 00
6028 00 00 00 00 00 00 00 00
6030 00 00 00 00 00 00 00 00
6038 00 00 00 00 00 00 00 00
*G5000……………………………………実行

*D6000                    ……実行結果
                          $37＝55
6000 01 02 03 04 05 06 07 08
6008 09 0A 37 00 00 00 00 00
6010 00 00 00 00 00 00 00 00
6018 00 00 00 00 00 00 00 00
6020 00 00 00 00 00 00 00 00
6028 00 00 00 00 00 00 00 00
6030 00 00 00 00 00 00 00 00
6038 00 00 00 00 00 00 00 00
*■

Break
Ready
```

〔図７−27　オートインクリメントのサンプルプログラム〕

```
 1  5000                    ORG   $5000
 2  5000 8E  6000           LDX   #$6000 ……………$6000番地に指標をつける
 3  5003 C6  0A             LDB   #10……………………ループカウンタをセット
 4  5005 4F                 CLRA……………………………合計をクリアしておく
 5  5006 AB  80     LOOP    ADDA  ,X+ ………………加算して，Xレジスタを１だけ増す
 6  5008 5A                 DECB……………………………カウントダウン
 7  5009 26  FB             BNE   LOOP………………………ループする
 8  500B A7  84             STA   ,X…………………………結果を格納
 9  500D 7E  ABF4           JMP   $ABF4……………………モニタへ
10                          END

ERROR(S) DETECTED
```

図７－29　図７－27の実行の様子

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=5000 LDX  #$6000 10回ループ
CC=00 A=00 B=00 DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5003 LDB  #$0A ····クリア
CC=00 A=00 B=0A DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5005 CLRA
CC=04 A=00 B=0A DP=00 X=6000 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=00 A=01 B=0A DP=00 X=6001 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=00 A=01 B=09 DP=00 X=6001 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=00 A=01 B=09 DP=00 X=6001 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=00 A=03 B=09 DP=00 X=6002 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=00 A=03 B=08 DP=00 X=6002 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=00 A=03 B=08 DP=00 X=6002 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=00 A=06 B=08 DP=00 X=6003 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=00 A=06 B=07 DP=00 X=6003 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=00 A=06 B=07 DP=00 X=6003 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=00 A=0A B=07 DP=00 X=6004 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=00 A=0A B=06 DP=00 X=6004 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=00 A=0A B=06 DP=00 X=6004 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=00 A=0F B=06 DP=00 X=6005 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=00 A=0F B=05 DP=00 X=6005 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=00 A=0F B=05 DP=00 X=6005 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=20 A=15 B=05 DP=00 X=6006 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=20 A=15 B=04 DP=00 X=6006 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=20 A=15 B=04 DP=00 X=6006 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=00 A=1C B=04 DP=00 X=6007 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=00 A=1C B=03 DP=00 X=6007 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=00 A=1C B=03 DP=00 X=6007 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=20 A=24 B=03 DP=00 X=6008 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=20 A=24 B=02 DP=00 X=6008 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=20 A=24 B=02 DP=00 X=6008 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=00 A=2D B=02 DP=00 X=6009 Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=00 A=2D B=01 DP=00 X=6009 Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=00 A=2D B=01 DP=00 X=6009 Y=0000 S=C07F U=0000 N=00 P=5006 ADDA ,X+
CC=20 A=37 B=01 DP=00 X=600A Y=0000 S=C07F U=0000 N=00 P=5008 DECB
CC=24 A=37 B=00 DP=00 X=600A Y=0000 S=C07F U=0000 N=00 P=5009 BNE  $5006
CC=24 A=37 B=00 DP=00 X=600A Y=0000 S=C07F U=0000 N=00 P=500B STA  ,X
CC=20 A=37 B=00 DP=00 X=600A Y=0000 S=C07F U=0000 N=00 P=500D JMP  $ABF4
REF-MP TRAP AT 500D
**
```

10回ループする

Xレジスタが1ずつ増しています.

〔図７－30　エクステンデット・インダイレクトのサンプル〕

```
  1    5000                         ORG    $5000
  2    5000 A6    9F 6000           LDA    [$6000]
  3    5004 E6    9F 6002           LDB    [$6002]
  4    5008 3D                      MUL
  5    5009 ED    9F 6004           STD    [$6004]
  6    500D 7E    ABF4              JMP    $ABF4
  7                                 END

0 ERROR(S) DETECTED
```

〔図７－32　インダイレクトの使用例〕

```
1    5000                      ORG    $5000
2    5000 8E    6000           LDX    #$6000
3    5003 A6    94             LDA    [,X]
4    5005 E6    98 02          LDB    [2,X]
5    5008 3D                   MUL
6    5009 ED    98 04          STD    [4,X]
7    500C 7E    ABF4           JMP    $ABF4
8                              END
```

うち左側の「インダイレクトでない場合」をみてきました。ここで右側の「**インダイレクトの場合**」について解説します。まずは、最下段の**エクステンデッドインダイレクト**について解説します。図7—30をみてください。これはこのエクステンデッドインダイレクトを用いたサンプルプログラムです。

2行めにエクステンドモードの場合とよく似た表記がオペランドにみられます。このように“［”と“］”で囲むとインダイレクトになります。それでは、この命令の動作をただのエクステンドモードと比較してみます。図7—31①のただのエクステンドモードの場合、オペランドの番地（＄６０００番地）が直接参照されます（**直接指定**といいます）。それに対して、エクステンデッドインダイレクトの場合には、オペランドの番地とその次の番地のメモリの内容をアドレスとしたメモリを参照します。このようにワンクッション置いて参照することを**間接指定**といいます。（図7—31②参照）。

次に図7—32のプログラムをみてください。これは図7—30のプログラムを、定数オフセットのインダイレクトを用いて書き直したものです。この場合も同様に、インダイレクトでないとしたときに対象となるアドレスと、その次のアドレスの内容をアドレスとするメモリを対象とします。

このように定数オフセットや他のオフセットには、インダイレクトでない場合に加えて、インダイレクトの場合の命令も持っています。ただし、5ビットオフセットとオートインクリメント（＋1）、オートデクリメント（－1）には、インダイレクトはありません。

図7—33をみてください。これは図7—32のプログラムを実行してみた様子です。図7—30のプログラムでも全く同じ結果が得られるはずですからやってみてください。

14. ポジションインディペンデント

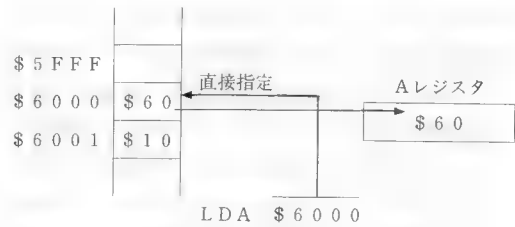
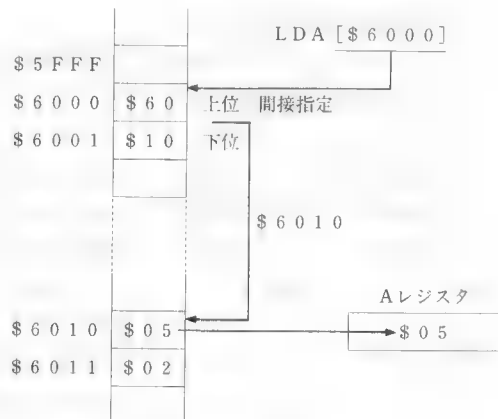
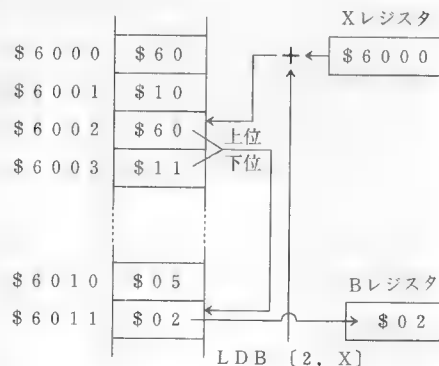
この節では、ちょっとインデックスモードを離れてポジションインディペンデント（位置独立）

① ただのエクステンドモードの場合

② エクステンデッドインダイレクトの場合

③ 定数オフセットのインダイレクトの場合

図7−31　インダイレクト

ということについて考えてみます。

だいぶ前に戻ってしまうことになりますが，図6−7をみてください。このプログラムは＄５０００番地から配置されていますが、なにかの都合で＄４０００番地から配置しなければならなくなったとしたらどうすればよいでしょうか。まず考えられるのは、１行めの格納番地の指定を＄５０００から＄４０００に書き換えて再度アセンブルする方法です。つまり＄４０００番地からの専用のプログラムを新しく作ってしまうというわけです。しかしいちいちアセンブルし直していたのでは不便です。＄５０００番地からの図6−7のプログラム（マシンコード）をそのまま＄４０００番地に移すだけで実行できたら便利です。このようにプログラム（マシンコード）を書き換えずに格納番地をかえることを「再配置する（リロケートする）」といいます。

図7—35は、図6−7をそのまま＄４０００番地からに再配置したものを逆アセンブルしたものです。逆アセンブルというのはメモリに格納されているマシンコードを読み取ってアセンブリ言語に直すことで、ちょうどアセンブルの逆になります。（付録に逆アセンブルを自動的に行う逆アセンブラのリストをのせてあります）。これをみると３箇所不都合なところがあることがわかります。ＪＳＲ命令の分岐先が、プログラムのない＄５０１Ｂ

〔図７−33　図７−32のダイプリント〕

〔図７−34　図７−32の実行例〕

〔図７−35　図７−34の再配置〕

番地になっています。正常に動作するためには、＄４０１Ｂでなければなりません。なぜこのようなことが生じるのかというと、サブルーチンへ分岐するのにJSR命令という**絶対分岐の命令**を使ったことによります。

次に図7—36をみてください。これはJSR命令ではなくBSR命令をつかったプログラムです。そして図7—37がこれを＄４０００番地から再配置したものを逆アセンブルしたものです。今度は不都合なところはなく、BSR命令の分岐先はちゃんと＄４０１８番地になっています。ですからBSR命令という**相対分岐の命令**を用いると不都合はおきなくなるわけです。このように、再配置可能なプログラムのことを**ポジションインディペンデント（位置独立）**なプログラムとか**リロケータブル**なプログラムといいます。

なぜリロケータブルであることが望まれるのかというと、リロケータブルであればどこにでも配置できるのでメモリ上に複数のプログラムを置いておくことができたり便利なことが多いからです。

〔図７－36　ＢＳＲ命令を使った例〕

```
 1   5000                      ORG    $5000
 2   5000 86   64              LDA    #100
 3   5002 8D   14              BSR    HANTEI
 4   5004 B7   6000            STA    $6000
 5   5007 86   3C              LDA    #60
 6   5009 8D   0D              BSR    HANTEI
 7   500B B7   6001            STA    $6001
 8   500E 86   00              LDA    #0
 9   5010 8D   06              BSR    HANTEI
10   5012 B7   6002            STA    $6002
11   5015 7E   ABF4            JMP    $ABF4
12
13   5018 81   3C     HANTEI   CMPA   #60
14   501A 25   04              BLO    HIKUI
15   501C 86   01              LDA    #1
16   501E 20   02              BRA    OWARI
17   5020 86   00     HIKUI    LDA    #0
18   5022 39          OWARI    RTS
19
20                             END

 0 ERROR(S) DETECTED
```


〔図７－37　図７－36再配置〕

```
4000 86 64      LDA   #$64
4002 8D 14      BSR   $4018
4004 B7 60 00   STA   $6000
4007 86 3C      LDA   #$3C
4009 8D 0D      BSR   $4018
400B B7 60 01   STA   $6001
400E 86 00      LDA   #$00
4010 8D 06      BSR   $4018
4012 B7 60 02   STA   $6002
4015 7E AB F4   JMP   $ABF4
4018 81 3C      CMPA  #$3C
401A 25 04      BCS   $4020
401C 86 01      LDA   #$01
401E 20 02      BRA   $4022
4020 86 00      LDA   #$00
4022 39         RTS
```

さて、ここで誤解のないように1つだけ注意をしておきます。先ほどの例では、絶対分岐命令を相対分岐命令にすればリロケータブルになることを示しましたが、これは絶対分岐命令を1つでも使用するとリロケータブルでなくなるというわけではありません。よく図7—36の11行めをみて絶対分岐命令を使っているからリロケータブルでないと言う人がいますがこれは間違いです。絶対分岐命令ではなく相対分岐命令でなければならないのは同一のプログラム内への分岐のときで、この例のようにモニタなどの位置が変わることのないアドレスに対して分岐する場合には、絶対分岐命令でなければいけません。もし、ここに相対分岐命令のLBRA命令を用いたとすると、リロケートしたときに＄ABF4番地に分岐しなくなってしまいます。このように、リロケータブルなプログラムを作るにはある意味で神経を使わなければいけませんが、リロケータブルであることは、それ以上の利益がありますから、これからはリロケータブルであることすなわちポジションインディペンデント（位置独立）であることを目指してプログラムを作るようにしてください。

15. PCリラティブ

さて、前節の図7—35のプログラムはリロケータブルではありましたが、1つだけ欠点があります。というのは、図7—35のプログラムはプログラム本体はリロケートできるけれども、データ領域は＄6000番地からに固体されているということです。ですからこのプログラムは＄6000番地付近に配置することはできません。なぜなら自分自身を書き換えてしまうかもしれないからです。

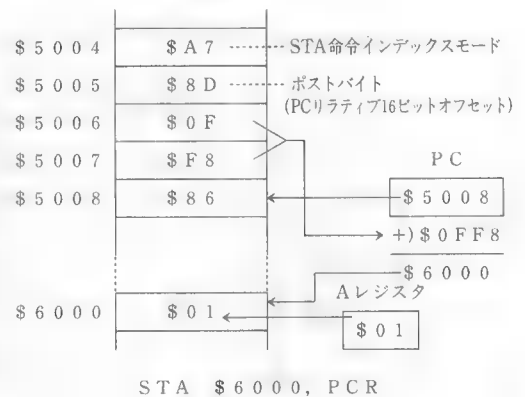



図7－39 PCリラティブの動作

〔図7－38 PCリラティブの使用例〕

```
 1   5000                          ORG    $5000
 2   5000 86   64                  LDA    #100
 3   5002 8D   17                  BSR    HANTEI
 4   5004 A7   8D OFF8             STA    $6000,PCR
 5   5008 86   3C                  LDA    #60
 6   500A 8D   0F                  BSR    HANTEI
 7   500C A7   8D OFF1             STA    $6001,PCR
 8   5010 86   00                  LDA    #0
 9   5012 8D   07                  BSR    HANTEI
10   5014 A7   8D OFEA             STA    $6002,PCR
11   5018 7E   ABF4                JMP    $ABF4
12
13   501B 81   3C        HANTEI    CMPA   #60
14   501D 25   04                  BLO    HIKUI
15   501F 86   01                  LDA    #1
16   5021 20   02                  BRA    OWARI
17   5023 86   00        HIKUI     LDA    #0
18   5025 39             OWARI     RTS
19
20                                 END

0 ERROR(S) DETECTED
```

（4・7・10行め）……PCリラティブを用いてデータ領域もリロケートさせる

これを避けるためには、プログラムをリロケートすればデータ領域もリロケートされるようにすることが望まれます。このためには、データ領域を指定するのに、絶対指定ではなく相対指定で行えばいいわけです。そして、これを実現するのがインデックスモード中のPCリラティブモードです。図7—38はこのPCリラティブを用いて図7—35を書き直したものです。違いはといえば、絶対指定であったところに“,PCR”がついただけですが、これで相対指定になっています。リラティブ（相対）ですのでブランチ命令と同様な過程を経てマシンコードが得られます。

それではプログラムの動作を追ってみます。2、3行めはすでに何度も出ているので省略して、4行めを解説します。4行めの命令を読み終えるとPCは$5008になり、命令の動作を開始します。ここでCPUはPCの値に、マシンコードのオペランドにかかれた値を加えて、$6000を得て、$6000番地にAレジスタをストアします。このようにPCリラティブは、アドレスを指定するのに、ブランチ命令と同様にPCの値からのへだたりで指定するので相対指定ができるというわけです。

さて、ここで一つ奇妙と思われる人もいるかもしれない点について解説します。今までのオフセット命令では

STA 1,X

などの場合、指標レジスタにオペランドの値を加えたアドレスが演算の対象になっていました（このように演算の対象となるアドレスのことを実効アドレスといいます）。この原則からいくと、

STA $6000,PCR

は、PCの値＋$6000が実効アドレスとなるようにみえます。たしかに中途半端なアセンブラの場合には、このように解釈してしまうものもありますがこれは間違いです（マシン語の解説書の中にもこのように解釈しているものもあって誤解を生んでいるようです）。

アセンブリ言語では、PCリラティブの場合には、実効アドレスとなるべき値（またはラベル）を書くことになっています。そして、アセンブラ（富士通やFLEXなどの正確なアセンブラ）は、実効アドレスとその時のPCの値とを計算してオペランドを決定します。PCリラティブを表わす“PCR”の“R”はこのことを示しています。だからといって図7—38の4行めを

STA $0FF8,PC

としても、エラーとなるか、“PCR”の略とみなしてアセンブルされてしまいます。この辺のことは、アセンブラのマニュアルを参照して確認しておくとよいでしょう。

このPCリラティブモードを用いてプログラムを組めば、データ領域を含めて完全にリロケータブルなプログラムを作成できます。6809CPUでプログラムを組むときには、ポジションインディペンデント（リロケータブル）であることは、実行速度が若干落ちるというデメリットがありますが、それ以上に利益があります。それに6809のプロのプログラマーの間では特殊な場合を除いてリロケータブルであることは常識となっていますので、PCリラティブモードをしっかり理解しておくことは重要です。

試しに図7—38のプログラムを$4000番地からにリロケートして実行してみたものを図7—40に示します。データ領域もリロケートに伴って、$5000番地からになっているのがわかります。

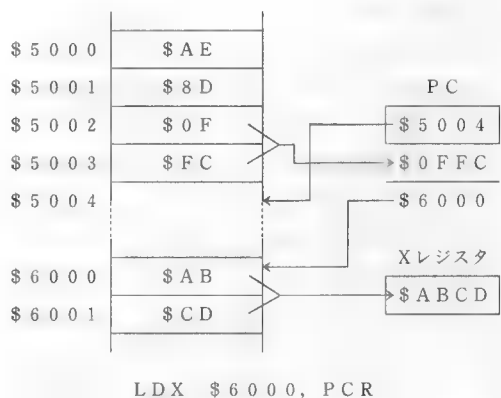
〔図7—40 図7—38の実行例〕

```
*D4000 ························ $4000番地からに再配置

4000 86 64 8D 17 A7 8D 0F F8
4008 86 3C 8D 0F A7 8D 0F F1
4010 86 00 8D 07 A7 8D 0F EA
4018 7E AB F4 81 3C 25 04 86
4020 01 20 02 86 00 39 00 00
4028 00 00 00 00 00 00 00 00
4030 00 00 00 00 00 00 00 00
4038 00 00 00 00 00 00 00 02
*G4000 ··········データ領域は$5000番地からに移動

*M5000 ······························································実行
5000 01-
5001 01-
5002 00-■
```

16. *LEA命令*

ポジションインディペンデントなプログラムを作成するのに必要欠くべからざる命令が１つあります。それがLEA命令です。この命令は、特殊な命令で若干わかりにくい部分があるので、LD命令と比較して解説しましょう。

図7—41は

LDA $6000, PCR

の動作を図示したものです。この図のLD命令というのは、オペランドで指定したアドレスのメモリの内容を指定したレジスタに格納する命令でした。次に図7—42をみてください。これは

LEAX $6000, PCR

の動作を図示したものです。この図にあるように、**LEA命令というのは、オペランドで指定したアドレスの値（指定したアドレスのメモリの内容ではなくアドレスの値そのもの）を指定したレジスタに格納する命令です。**この例の場合、オペラン

図7－41 LD命令の動作

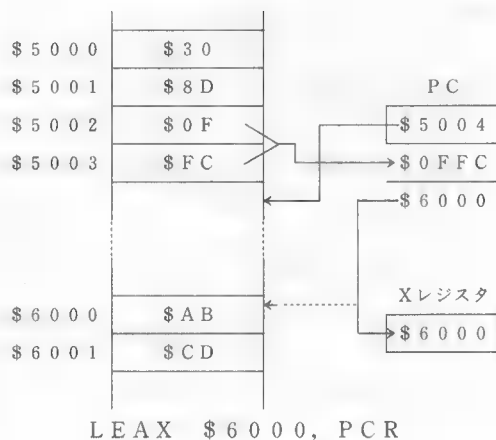



図7－42 LEA命令の動作

〔図７－43 ＬＥＡ命令のサンプルプログラム〕

```
 1   5000                         ORG   $5000
 2   5000 30   8D OFFC            LEAX  $6000,PCR ………ＬＥＡ命令を用いて
 3   5004 86   64                 LDA   #100             ＄６０００番地に指標をつける.
 4   5006 8D   11                 BSR   HANTEI
 5   5008 A7   84                 STA   ,X
 6   500A 86   3C                 LDA   #60
 7   500C 8D   0B                 BSR   HANTEI
 8   500E A7   01                 STA   1,X
 9   5010 86   00                 LDA   #0
10   5012 8D   05                 BSR   HANTEI
11   5014 A7   02                 STA   2,X
12   5016 7E   ABF4               JMP   $ABF4
13
14   5019 81   3C        HANTEI   CMPA  #60
15   501B 25   04                 BLO   HIKUI
16   501D 86   01                 LDA   #1
17   501F 20   02                 BRA   OWARI
18   5021 86   00        HIKUI    LDA   #0
19   5023 39             OWARI    RTS
20
21                                END

0 ERROR(S) DETECTED
```

ドは＄６０００番地を指定しているのでＸレジスタに＄６０００が代入されます。LEA命令は、インデックスモードの実効アドレス値をそのまま指定したレジスタに代入する命令であるわけです（LEA命令にはインデックスモードしかありません)。それでは、このLEA命令はどのような場合に用いるのでしょうか。

インデックスモードを用いてプログラムを作る場合、まず最初に指標をつけなければなりませんでした。これまでは

LDX #$6000

などとやってきましたが、これではリロケートしてもデータ領域(すなわち指標をつけるアドレス)は動いてくれません。そこでこのような場合に相対指定をするのですが、

LDX #$6000, PCR

などというわけにはいかないので、このＬＥＡ命令を用いて

LEAX $6000, PCR

とします。こうすれば、プログラムはリロケートすれば、指標を付けるアドレスは相対指定になっているのでデータ領域の方もリロケートされます。

これらを用いた例を図7―43にあげます。2行めでこのLEA命令を用いています。これにより、プログラムがもし＄４０００番地から配置されたとすれば、Ｘレジスタには＄５０００が代入されデータ領域は＄５０００番地からに配置されます。その実行例を図7―44に示します。

〔図7－44　図7－43の実行例〕

```
*D4000··························＄４０００番地からに再配置

4000 30 8D OF FC 86 64 8D 11
4008 A7 84 86 3C 8D 0B A7 01
4010 86 00 8D 05 A7 02 7E AB
4018 F4 81 3C 25 04 86 01 20
4020 02 86 Q0 39 00 00 00 00
4028 00 00 00 00 00 00 00 00
4030 00 00 00 00 00 00 00 00
4038 00 00 00 00 00 00 00 02
*G4000·················································実行

*M5000···········データ領域は＄５０００番地からに移動
5000 01-
5001 01-
5002 00-■
```

17. アドレッシングモードのまとめ

さて、長い第７章もこの節が最後になります。この７章ではアドレッシングモードを中心に学習してきましたが前節までで、そのすべてを学習したことになります。そこで、この節ではそのまとめとして、全アドレッシングモードの一覧表をあげておくことにします。それぞれをよくみて意味が理解できているかどうか確認して、もし不十分な箇所があればそのモードを再復習しておいてください。

アドレッシングモード		解　　説		オペランド形式	例
イミディエイト・モード		オペランドを定数とみなす.		#$nn	LDA #$AA
ダイレクト・モード		上位8ビットにDPレジスタを，下位8ビットにオペランドの値を用いてアドレスとし，そのアドレスのメモリを対象とする		<$nn	LDA <$AA
エクステンド・モード		オペランドの示すアドレスのメモリを対象とする		>$nnnn $nnnn	LDA $5678
インヘレント・モード		オペランドを必要としない			CLRA
リラティブ・モード		相対分岐に用いる		$nnnn	BRA $nnnn
インデックス・モード	ゼロオフセット	レジスタRの内容を実効アドレスとする	D	，R	，X
			I	〔，R〕	〔，X〕
	5ビットオフセット	R＋nを実効アドレスとする（nは5ビットの補数表現）	D	n，R	−2，Y
			I	なし	なし
	8ビットオフセット	R＋nを実効アドレスとする（nは8ビットの補数表現）	D	n，R	−100，X
			I	〔n，R〕	〔+65，Y〕
	16ビットオフセット	R＋nnを実効アドレスとする（nnは16ビットの補数表現）	D	nn，R	10000，X
			I	〔nn，R〕	〔$1234，Y〕
	Aレジスタオフセット	R＋Aを実効アドレスとする（Aレジスタは補数表現）	D	A，R	A，Y
			I	〔A，R〕	〔A，X〕
	Bレジスタオフセット	R＋Bを実効アドレスとする（Bレジスタは補数表現）	D	B，R	B，X
			I	〔B，R〕	〔B，U〕
	Dレジスタオフセット	R＋Dを実効アドレスとする（Dレジスタは補数表現）	D	D，R	D，U
			I	〔D，R〕	〔D，Y〕
	オートインクリメント ＋1	演算後，R←R＋1	D	，R＋	，X＋
			I	なし	なし
	オートインクリメント ＋2	演算後，R←R＋2	D	，R＋＋	，S＋＋
			I	〔，R＋＋〕	〔，S＋＋〕
	オートデクリメント −1	演算前に，R←R−1	D	，−R	，−S
			I	なし	
	オートデクリメント −2	演算前に，R←R−2	D	，−−R	，−−U
			I	〔，−−R〕	〔，−−X〕
	PCリラティブ 8ビットオフセット	PC相対で実効アドレスを決定	D	n，PCR	$6000，PCR
			I	〔n，PCR〕	〔$5000，PCR〕
	PCリラティブ 16ビットオフセット	PC相対で実効アドレスを決定	D	nn，PCR	$6000，PCR
			I	〔nn，PCR〕	〔$6000，PCR〕
	エクステンデッド インダイレクト	オペランドの示すアドレスの内容を実効アドレスとする	I	〔$nnnn〕	〔$FBFA〕

・インデックスモードの例にはオペランドのみを示した.
・D：インダイレクトでない場合，I：インダイレクトの場合，解説はインダイレクトでない場合について行なってあるので注意.

図7−45　アドレッシングモード一覧表

第8章

マシン語プログラミング part Ⅵ

——論理演算と残りの命令——

1. 論理演算

これまでの演算命令は主に算術演算を扱ってきましたが、ここで、ちょっと毛色の違った論理演算を扱います。

この論理演算があるおかげでＣＰＵは複雑な条件がからみあった判断が実現できるという点で重要です。6809ＣＰＵには、全部で４種の論理演算命令があります。

◀ＡＮＤ命令（logical and）▶

日本語の「かつ」に相当する論理積を取る命令です。図8—1に真理値表を示します。この命令は

ANDA #$CD

のように、加算命令と同様にして用いますが、指定できるレジスタは、ＡレジスタとＢレジスタそれにＣＣレジスタに限られます。さらにＣＣレジスタの場合には、アドレッシングモードがイミディエイトモードに限られます。図8—2にＡＮＤ演算の例を示しますが、この様にビットごとにＡＮＤが取られて結果となります。

◀ＯＲ命令（inclusive or）▶

日本語の「または」に相当する論理和を取る命令です。図8—3に真理値表、図8—4にＯＲ演算の例を示します。用い方はＡＮＤ命令と同じです。

◀ＥＯＲ命令（exclusive or）▶

日本語に相当する語はありませんが、排他的論理和を取る命令です。図8—5は真理値表、図8—6はＥＯＲ演算の例です。使い方はＡＮＤ命令と同様ですが、ＣＣレジスタは指定できません（こちらのＥＯＲの方を日本語の「または」として用いることもあるようです）。

◀ＣＯＭ命令（complement）▶

日本語の「～でない」に相当する否定を取る命令です。この命令は他の論理演算命令が２項演算（２つの値から１つの結果を得る演算）であるのに対して、単項演算（１つの値から１つの結果を得る演算）になっています。図8—7は真理値表、

		AND
0	0	0
0	1	0
1	0	0
1	1	1

図8－1　AND演算の真理値表

```
    1010 1011 $AB
AND 1100 1101 $CD
-----------------
    1000 1001 $89
```

図8－2　AND演算の例

		OR
0	0	0
0	1	1
1	0	1
1	1	1

図8－3　OR演算の真理値表

```
    1010 1011 $AB
OR  1100 1101 $CD
-----------------
    1110 1111 $EF
```

図8－4　OR演算の例

		EOR
0	0	0
0	1	1
1	0	1
1	1	0

図8－5　EOR演算の真理値表

```
    1010 1011 $AB
EOR 1100 1101 $CD
-----------------
    0110 0110 $66
```

図8－6　EOR演算の例

	COM
0	1
1	0

図8－7　COM演算の真理値表

```
COM 1010 1011 $AB
-----------------
    0101 0100 $54
```

図8－8　COM演算の例

〔図8－9　論理演算のサンプルプログラム〕

```
  1  5000                    ORG   $5000
  2  5000 30    8D OFFC      LEAX  $6000,PCR………ポジションインディペンデント
                                                  にする
  3  5004 A6    84           LDA   ,X
  4  5006 A4    01           ANDA  1,X ┐
  5  5008 A7    02           STA   2,X ┘……………………ANDを（X＋2）番地へ
  6  500A A6    84           LDA   ,X
  7  500C AA    01           ORA   1,X ┐
  8  500E A7    03           STA   3,X ┘……………………ORを（X＋3）番地へ
  9  5010 A6    84           LDA   ,X
 10  5012 A8    01           EORA  1,X ┐
 11  5014 A7    04           STA   4,X ┘……………………EORを（X＋4）番地へ
 12  5016 A6    84           LDA   ,X
 13  5018 43                 COMA      ┐
 14  5019 A7    05           STA   5,X ┘……………………COMを（X＋5）番地へ
 15  501B 7E    ABF4         JMP   $ABF4
 16                          END

O ERROR(S) DETECTED
```

図8―8はCOM演算の例を示します。COM命令は単項演算ですから、

COMA　　,COM $6000

などというようにCLR，DEC命令などと同様にして用います。

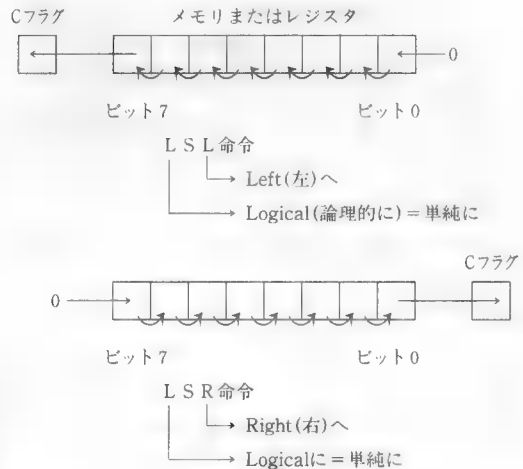
このように4種の論理演算命令があるわけですが、図8―9でここに示した演算例を確認するプログラムを示し、図8―10にその実行例を示します。それぞれの動作を確認してください。

〔図8－10　図8－9の実行例〕

```
*D4000……$4000番地からにリロケートとして実行
                                        させてみる
4000 30 8D 0F FC A6 84 A4 01
4008 A7 02 A6 84 AA 01 A7 03
4010 A6 84 A8 01 A7 04 A6 84
4018 43 A7 05 7E AB F4 00 00
4020 00 00 00 00 00 00 00 00
4028 00 00 00 00 00 00 00 00
4030 00 00 00 00 00 00 00 00
4038 00 00 00 00 00 00 00 00
*■

*M5000………………データ領域も$5000番地に移動
5000 00-AB ┐
5001 00-CD ┘…………………………データをセット
5002 00-
5003 00-
5004 00-
5005 00-.

*G4000……………………………$4000から実行

*M5000
5000 AB-
5001 CD-
5002 89-……………………………………AND
5003 EF-……………………………………OR
5004 66-……………………………………EOR
5005 54-……………………………………COM
5006 00-.

*■
```

2. シフトローテート命令

前節の論理演算命令は各ビットに対して演算を行う細い命令でしたが、ここで解説する命令もある意味で細い命令といえて、メモリあるいはレジ

図8－12　LSL・LSR命令の動作

〔図8－11　図8－9の実行の様子（$4000番地から）〕

```
                                  ┌……………………Xレジスタもリロケートに伴って変化している
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 LEAX $5000,PCR
CC=00 A=00 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4004 LDA  ,X
CC=08 A=AB B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4006 ANDA $1,X
CC=08 A=89 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4008 STA  $2,X
CC=08 A=89 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400A LDA  ,X
CC=08 A=AB B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400C ORA  $1,X
CC=08 A=EF B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400E STA  $3,X
CC=08 A=EF B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4010 LDA  ,X
CC=08 A=AB B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4012 EORA $1,X
CC=00 A=66 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4014 STA  $4,X
CC=00 A=66 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4016 LDA  ,X
CC=08 A=AB B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4018 COMA
CC=01 A=54 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4019 STA  $5,X
CC=01 A=54 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=401B JMP  $ABF4
REF-MP TRAP AT 401B
**
```

スタの各ビットを右や左にずらす命令群です。同じような命令が続きますが、うまく分類されて名前がつけられているので理解しやすいと思います。

◀ＬＳＬ命令・ＬＳＲ命令▶

この命令は、メモリまたはレジスタ内のデータを単純に右や左にずらす命令です。図8—12をみればその動作は一目瞭然でしょう。すなわち指定された方向（ＬかＲ）に各ビットをずらすわけです。そして、空いたビット（LSLならビット0、ＬSRならビット7）には0を入れ、あふれたビット（LSLなら元のビット7、LSRなら元のビット0）は、ＣＣレジスタ内のＣフラグ（キャリーフラグ）に格納します。

さて、これらの命令の意味ですが単純にずらす（Shift）するだけではありません。実際にやってみるとよくわかりますが、これらの命令の実行によってメモリまたはレジスタの内容は、ＬＳＬ命令ならば2倍、ＬＳＲ命令だと½倍されるということがあります。実際にその実験をしたものを図8—13に示します。

◀ＡＳＬ命令・ＡＳＲ命令▶

前に示したLSL・LSRは数を2倍、½倍にするものでしたが、扱う数は絶対値表現のものに限られています。そこで補数表現の数に対して、2倍、½倍を実行しようというのがASL命令と

〔図8－14　図8－13の実行例〕

```
*D4000··························$4000番地にリロケート

4000 30 8D 0F FC A6 84 48 A7
4008 01 A6 84 44 A7 02 7E AB
4010 F4 A7 03 A6 84 47 A7 04
4018 7E AB F4 00 00 00 00 00
4020 00 00 00 00 00 00 00 00
4028 00 00 00 00 00 00 00 00
4030 00 00 00 00 00 00 00 00
4038 00 00 00 00 00 00 00 00
*M5000
5000 00-66······················$66＝102をセット
5001 00-.

*G4000

*M5000
5000 66-
5001 CC-································$CC＝204
5002 33-■·······························$33＝ 51
```

〔図8－13　ＬＳＬ・ＬＳＲ命令〕

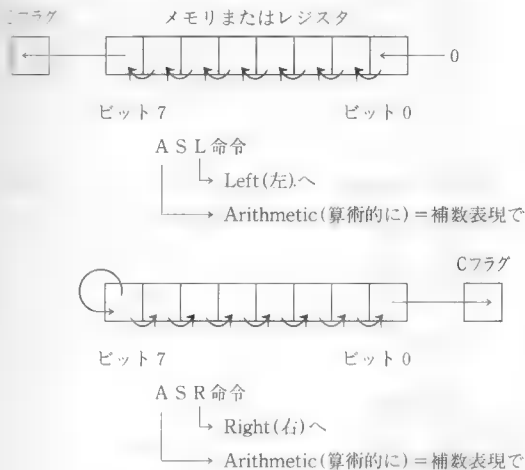
```
 1   5000                    ORG    $5000
 2   5000 30    8D OFFC      LEAX   $6000,PCR
 3   5004 A6    84           LDA    ,X
 4   5006 48                 LSLA·····························左へシフト（＊2）
 5   5007 A7    01           STA    1,X
 6   5009 A6    84           LDA    ,X
 7   500B 44                 LSRA·····························右へシフト（＊½）
 8   500C A7    02           STA    2,X
 9   500E 7E    ABF4         JMP    $ABF4
10                           END

 0 ERROR(S) DETECTED
```

〔図8－15　図8－13の実行の様子〕

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 LEAX $5000,PCR
CC=00 A=00 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4004 LDA  ,X
CC=00 A=66 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4006 ASLA
CC=0A A=CC B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4007 STA  $1,X
CC=08 A=CC B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4009 LDA  ,X
CC=00 A=66 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400B LSRA
CC=00 A=33 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400C STA  $2,X
CC=00 A=33 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400E JMP  $ABF4
REF-MP TRAP AT 400E
```

ASR 命令です。その動作は図8−16に示すとおりです。注目すべきなのは ASL 命令の動作が LSL 命令と全く同じということです。そのため、マシンコードは、ASL 命令と LSL 命令では同一になっています。一方、ASR 命令は LSR 命令とは違って常にビット 7 が保存されます。

図8−17〜図8−19はこれらの命令の実験例です。

図8−16　ASL・ASR命令の動作

〔図8−18Ⓑ図8−17の実行例〕

```
*D4000·························$4000番地にリロケート

4000 30 8D 0F FC A6 84 48 A7
4008 01 A6 84 47 A7 02 7E AB
4010 F4 00 00 00 00 00 00 00
4018 00 00 00 00 00 00 00 00
4020 00 00 00 00 00 00 00 00
4028 00 00 00 00 00 00 00 00
4030 00 00 00 00 00 00 00 00
4038 00 00 00 00 00 00 00 00
*M5000
5000 00-F6·····························−10をセット
5001 00-.

*G4000··········································実行

*M5000
5000 F6-
5001 EC- ·······························−20＝−10×2
5002 FB-■·····························−  5＝−10×½
```

〔図8−17　ASL・ASR命令〕

```
 1   5000                ORG    $5000
 2   5000 30   8D 0FFC   LEAX   $6000,PCR
 3   5004 A6   84        LDA    ,X
 4   5006 48             ASLA··························2倍する
 5   5007 A7   01        STA    1,X
 6   5009 A6   84        LDA    ,X
 7   500B 47             ASRA··························½倍する
 8   500C A7   02        STA    2,X
 9   500E 7E   ABF4      JMP    $ABF4
10                       END

0 ERROR(S) DETECTED
```

〔図8−18Ⓐ図8−17の実行の様子〕

```
     ┌─CフラグON
CC=00│A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 LEAX $5000,PCR
CC=00│A=00 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4004 LDA  ,X
CC=08│A=F6 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4006 ASLA
CC=[09] A=EC B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4007 STA  $1,X
CC=09 A=EC B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4009 LDA  ,X
CC=09 A=F6 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400B ASRA
CC=08 A=FB B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400C STA  $2,X
CC=08 A=FB B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400E JMP  $ABF4
REF-MP TRAP AT 400E
```

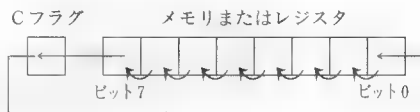



図8－20　ROL・ROR命令の動作

〔図8－23　図8－22の実行例〕

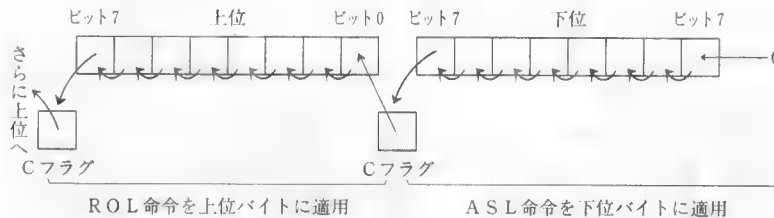
```
*D4000 ························$4000からにリロケート

4000 30 8D 0F FC EC 84 44 56
4008 ED 02 EC 84 47 56 ED 04
4010 7E AB F4 00 00 00 00 00
4018 00 00 00 00 00 00 00 00
4020 00 00 00 00 00 00 00 00
4028 00 00 00 00 00 00 00 00
4030 00 00 00 00 00 00 00 00
4038 00 00 00 00 00 00 00 00
*■

*M5000
5000 00-FD ┐····$FDA8 ｛絶対値表現で64936
5001 00-A8 ┘           ｛補数表現で－600
5002 00-
5003 00-
5004 00-
5005 00-.

*G4000 ··································実行

*M5000
5000 FD-
5001 A8-
5002 7E- ┐························絶対値表現：32468
5003 D4- ┘
5004 FE- ┐····························補数表現：－300
5005 D4-■┘
```

ASL　下位
ROL　上位
必要ならば（ROL　さらに上位
　　　　　　　⋮　　　　　　　　　の順で実行する

図8－21　ASL命令の拡張

〔図8－22　ROR命令による2バイトのシフト〕

```
 1    5000                     ORG    $5000
 2    5000 30     8D OFFC      LEAX   $6000,PCR
 3    5004 EC     84           LDD    ,X
 4    5006 44                  LSRA ┐········論理的シフト（絶対値表現で½倍） 上位のシフト
 5    5007 56                  RORB ┘                                         下位のシフト
 6    5008 ED     02           STD    2,X
 7    500A EC     84           LDD    ,X
 8    500C 47                  ASRA ┐··········算術的シフト（補数表現で½倍） 上位のシフト
 9    500D 56                  RORB ┘                                        下位のシフト
10    500E ED     04           STD    4,X
11    5010 7E     ABF4         JMP    $ABF4
12                             END

0 ERROR(S) DETECTED
```

◀ROL命令・ROR命令▶

これまでは1バイト単位でのシフト（ずらし）を見てきましたが、ROL，ROR命令を利用することにより2バイト以上のシフトも可能になります。まずは、ROL，ROR命令の動作を図8—20で確認してください。ともに、Cフラグを含めて移動（回転）していることがわかると思います。

それでは、この命令を用いてどのように2バイト以上のシフトを実現するのかを解説しましょう。図8—21をみてください。これは、ROL命令を用いてASL命令を拡張して2バイトに対して行なっている例です。つまり、キャリーフラグを中継点として、上位バイトのビット0と下位バイトのビット0とをつなげるわけです。

図8—22は、ROR命令を用いた拡張の例です。今度は右シフトなので上位から適用します。図8—23・図8—24はその実行例です。

3. 残りの命令

これまでに数多くの命令を学習してきましたが、いよいよ残る命令は、図8—25の10種だけになりました。しかし、このうち＊印をつけた4種は、割り込みという概念と関係していて、ちょっと難しいので除くことにします。とすると、残りは6種だけです。これからはこの残りの命令を1つずつ解説することにします。

〔図8－24　図8－22の実行の様子〕

Cフラグ＝1　下位バイトへの中継

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 LEAX $5000,PCR
CC=00 A=00 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4004 LDD  ,X
CC=08 A=FD B=A8 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4006 LSRA
CC=01 A=7E B=A8 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4007 RORB
CC=08 A=7E B=D4 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4008 STD  $2,X
CC=00 A=7E B=D4 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400A LDD  ,X
CC=08 A=FD B=A8 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400C ASRA
CC=09 A=FE B=A8 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400D RORB
CC=08 A=FE B=D4 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400E STD  $4,X
CC=08 A=FE B=D4 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4010 JMP  $ABF4
REF-MP TRAP AT 4010
**
```

〔図8－26　ABX命令の実行例〕

```
 1  5000              ORG  $5000
 2  5000 8E   6809    LDX  #$6809
 3  5003 C6   FA      LDB  #$FA
 4  5005 3A           ABX
 5  5006 AF   8D OFF6 STX  $6000,PCR
 6  500A 7E   ABF4    JMP  $ABF4
 7                    END

O ERROR(S) DETECTED
```

$6809＋$FAを求める

$4000番地にリロケートされている

```
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 LDX  #$6809
CC=00 A=00 B=00 DP=00 X=6809 Y=0000 S=C07F U=0000 N=00 P=4003 LDB  #$FA
CC=08 A=00 B=FA DP=00 X=6809 Y=0000 S=C07F U=0000 N=00 P=4005 ABX
CC=08 A=00 B=FA DP=00 X=6903 Y=0000 S=C07F U=0000 N=00 P=4006 STX  $5000,PCR
CC=00 A=00 B=FA DP=00 X=6903 Y=0000 S=C07F U=0000 N=00 P=400A JMP  $ABF4
REF-MP TRAP AT 400A
**
```

$6809＋$00FA＝$6903

◀ＡＢＸ命令▶

この命令は、ＸレジスタにＢレジスタの値を絶対値表現とみなして加算する命令です。この命令は特徴的で、加算を１バイトのインヘレントモードで実現でき、さらにＣＣレジスタの各フラグはこの命令によっては全く変化しません。

図8—26にこの命令の実行例を示します。

◀ＢＩＴ命令▶

この命令は、ＡレジスタまたはＢレジスタの内容とオペランドの示す値とでＡＮＤ（論理積）を取りその結果によりＣＣフラグを変化させます。ここまではＡＮＤ命令と同じですが、この命令は結果をレジスタに再格納しないので、レジスタの内容は保存されます。このため、 BIT命令は、

BITA #$08

というふうに用いて、Ａレジスタのビット単位の1/0検査などに用います。この例では、Ａレジスタのビット３が０ならＺフラグがセットされます。

図8—27にこの命令の実行例を示します。

◀ＴＳＴ命令▶

この命令は、レジスタやメモリが０であるか、または負であるかを調べる命令です。指定したメモリかレジスタの全ビットが“0”、つまり０ならば、Ｚフラグがセット（“1”）されます。また、補数表現とみなすと負である（ビット７が１）場合にはＮフラグがセットされます。

図8—28にこの命令の実行例を示します。＄６０００番地の内容が０ならば０を、０でないならば＄ＦＦを＄６００１番地に格納するプログラムです。

ＡＢＸ	ＣＷＡＩ*
ＴＳＴ	ＲＴＩ*
ＢＩＴ	ＳＷＩ*
ＤＡＡ	ＳＹＮＣ*
ＮＯＰ	
ＢＲＮ	

図8－25　未だ学習していない命令

〔図８－27　ＢＩＴ命令の実行例〕

```
 1    5000                            ORG    $5000
 2    5000 A6   8D OFFC               LDA    $6000,PCR
 3    5004 85   08                    BITA   #$08 ……………ビット3が1であるかをチェック
 4    5006 27   04                    BEQ    OFF……………………………0のときはジャンプ
 5    5008 86   FF                    LDA    #$FF
 6    500A 20   01                    BRA    STORE
 7    500C 4F               OFF       CLRA
 8    500D A7   8D OFFO     STORE     STA    $6001,PCR
 9    5011 7E   ABF4                  JMP    $ABF4
10                                    END

O ERROR(S) DETECTED                ON
                                   ↓
                        $69=%01101001のとき            $4000番地からにリロケートした
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 LDA  $5000,PCR
CC=00 A=69 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4004 BITA #$08
CC=00 A=69 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4006 BEQ  $400C
CC=00 A=69 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4008 LDA  #$FF
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400A BRA  $400D
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400D STA  $5001,PCR
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4011 JMP  $ABF4
REF-MP TRAP AT 4011                ←OFF
**                      $F7=%11110111のとき
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 LDA  $5000,PCR
CC=08 A=F7 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4004 BITA #$08
CC=04 A=F7 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4006 BEQ  $400C
CC=04 A=F7 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400C CLRA
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400D STA  $5001,PCR
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4011 JMP  $ABF4
REF-MP TRAP AT 4011
**
```

◀ＮＯＰ命令▶

この命令は、何もしない (No Operation) 命令です。つまりＣＰＵはこの命令を読み込んでも何もせずに次の命令の実行に移ります。

何もしないのでは役に立たないと思うかもしれませんが、ちゃんと用途はあります。例えば、バグ(bug：虫＝間違いのこと)がある未完成のプログラム中に NOP を散りばめておくと、途中で命令を追加したくなったときに、この NOP をつぶして新しい命令に置き換えることができます。また、既にある命令を NOP に置き換えることによって、その命令を除いてしまうこともできるわけです。こういった意味でプログラムの応急処置には有用な命令ですから、この NOP 命令のＯＰコード＄１２を覚えておくと便利です。

◀ＢＲＮ命令・ＬＢＲＮ命令▶

これらの命令も NOP 命令と同様の性格を持っています。これらはブランチ命令で、リラティブモードを用いるのですが、決して分岐しません。決して分岐しないものの分岐先はどこでも構いませんから、この命令のオペランドは意味を持ちません。

この命令は、プログラム中の（分岐する）ブランチ命令を無効にするのに用います。例えば図8—28のプログラムの３行目の BEQ 命令を無効にしたければ、＄５００４番地の＄２７を BRN 命令のＯＰコードである＄２１に書き換えればいいわけです。単に無効にするだけでしたら＄５００４番地と＄５００５番地に＄１２（NOP）を書き込んでもよいのですが、 BRN 命令を用いれば、オペランドを書き換えずにすむので、すぐに元に戻すことができます（＄５００４番地に＄２７を書き込めば元に戻ります)。

4. BCDコード

ＣＰＵは基本的に２進数しか扱うことはできません。ユーザにみやすいように置き換えたとしても16進数です。ところが、ＣＰＵは擬似的に10進数を扱えるようになっています。この節ではそのことに関して解説してみましょう。

16進数では０～９とＡ～Ｆの文字を使いますがＡ～Ｆを考えずに＄２１なら10進数の21を示すも

〔図８－28　ＴＳＴ命令の実行例〕

```
1    5000                           ORG    $5000
2    5000 6D    8D OFFC             TST    $6000,PCR ………$6000番地の内容を調べる
3    5004 27    04                  BEQ    ZERO ……………………………ゼロなら分岐
4    5006 86    FF                  LDA    #$FF
5    5008 20    01                  BRA    STORE
6    500A 4F             ZERO       CLRA
7    500B A7    8D OFF2  STORE      STA    $6001,PCR
8    500F 7E    ABF4                JMP    $ABF4
9                                   END

0 ERROR(S) DETECTED
                          $5000番地が0のとき                    $4000番地からにリロケートしてある
CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 TST  $5000,PCR
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4004 BEQ  $400A
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400A CLRA
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400B STA  $5001,PCR
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400F JMP  $ABF4
REF-MP TRAP AT 400F
**                        $5000番地が$FFのとき
CC=04 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 TST  $5000,PCR
CC=08 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4004 BEQ  $400A
CC=08 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4006 LDA  #$FF
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4008 BRA  $400B
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400B STA  $5001,PCR
CC=08 A=FF B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=400F JMP  $ABF4
REF-MP TRAP AT 400F
**        └─Nフラグが1になっている
```

のと考えると、私たち人間にとってわかりやすく非常に便利です。この方法は「２進化10進数(Binary Coded Decimal：ＢＣＤ)」と呼ばれています。この方法を用いれば、１バイトで０～99までの百個の数を表すことができます。

しかし、この方法にも弱点があります。68＋９のつもりで＄６８＋＄０９を実行します。すると得られる結果は(ＣＰＵは２進数で計算するので)＄７１となってしまい、68＋９≠71で答えが違ってしまいます。そこでつじつまを合わせるのがＤＡＡ命令です。

このＤＡＡ命令は加算命令の直後に用いて10進数に補正する命令です。詳しい動作は省略しますが、加算命令によってセットされたＣフラグやＨフラグを用いることでつじつまを合わせます（この命令がただ１つＨフラグを利用する命令です)。

図8―29がこのＤＡＡ命令を使用した例です。実行例では68＋09を実行していますが、ＤＡＡ命令によって＄７１が＄７７に補正されて正しい答68＋09＝77が得られています。

```
        $39  0011  1001
AND   #$0F  0000  1111   $0Fでマスク
─────────────────────
             0000  1001
             マスクされた
```

図8－30　AND命令によるマスク

残念なのは、この補正は減算や乗算命令の後では利用できないので、ＢＣＤ演算は加算しかできません。この点は6502などのＣＰＵに対して劣っている点でもあります。

5. もう一つの効用

前節までで割り込み関係の命令を除いて全ての命令を学習したわけですが、ここでは、それらの命令のうち既に述べた用途とちょっと違った使い方のできる命令について、若干解説したいと思います。

```
        $75  0111  0101
OR    #$08  0000  1000   ビット3をONにする
─────────────────────
             0111  1101
                   ↑
                   └ONになる
```

図8－31　OR命令の応用

```
        $39  0011  1001
EOR   #$08  0000  1000   ビット3を反転する
─────────────────────
             0011  0001
                   ↑
                   └反転している
```

図8－32　EOR命令の応用

〔図８－29　ＢＣＤによる演算〕

```
   1    5000                    ORG     $5000
   2    5000 30    8D OFFC      LEAX    $6000,PCR
   3    5004 A6    84           LDA     ,X
   4    5006 AB    01           ADDA    1,X··················加算
   5    5008 19                 DAA ··········２進化１０進数に補正
   6    5009 A7    02           STA     2,X
   7    500B 7E    ABF4         JMP     $ABF4
   8                            END

 0 ERROR(S) DETECTED

CC=00 A=00 B=00 DP=00 X=0000 Y=0000 S=C07F U=0000 N=00 P=4000 LEAX $5000,PCR
CC=00 A=00 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4004 LDA  ,X
CC=00 A=68 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4006 ADDA $1,X
CC=20 A=71 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4008 DAA
CC=20 A=77 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=4009 STA  $2,X
CC=20 A=77 B=00 DP=00 X=5000 Y=0000 S=C07F U=0000 N=00 P=400B JMP  $ABF4
REF-MP TRAP AT 400B       $68+$09=$71→77 (=68+9)
**
        └─補正されている
```

◀マスクとビットのＯＮ・ＯＦＦ▶

論理演算命令の解説では主に演算そのものについて述べましたが、論理演算命令はどちらかといえば、これから述べる用途に用いられる場合の方が多くなります。

まずAND命令はマスクという用途に用いられます。マスクとは、＄３９のうち下位4ビットだけがほしいときに＄０Ｆという値でANDを取ることによって＄０９を得るということです。

残しておきたいビットを1にした値でANDをとるとマスクできます（図8―30)。また逆に考えて0にしたいビットを0に他のビットを1にした値でANDを取れば、特定のビットをＯＦＦ(０)にすることもできます。

次に、OR命令は、特定のビットをＯＮ（１）にしたい場合に用います。ＯＮにしたいビットを1にした値でORをとることにより実現できます（図8―31)。

さらに、EOR命令は、特定のビットを反転させたい場合に用います。これは、反転させたいビットを1にした値でEORをとることにより実現できます（図8―32)。

◀16ビットの加算命令▶

これまで16ビットの加算といえばDレジスタを用いたものだけでした。しかし実は、Xレジスタなどのインデックス用レジスタでも加算命令を実行できるのです。それを実現するのはLEA命令です。

例えば、Xレジスタに1加算したいのであれば，

LEAX 1,X

とすればよいわけです。さらにAレジスタ・Bレジスタ・Dレジスタを含めて加算することもできます。例えば

LEAX B,X

とすれば、XレジスタにBレジスタの値が加算されます。既にXレジスタとBレジスタの加算にはABXという命令を学習しましたが、このLEA命令を用いた加算は、ABX命令がBレジスタを絶対値表現とみなしたのに対して、Bレジスタを補数表現とみなして加算するので注意が必要です。

また、

LEAU −10,X

のようにXレジスタの値から10引いた値をUレジスタに格納するというふうに使用することもできます。

ただし、このＬＥＡ命令を用いた加算には注意すべき点があります。というのは、LEAX，LEAY命令では、Ｚフラグしか変化しないということ、LEAU，LEAS命令ではＣＣレジスタ内のフラグは全く変化しないということです。

このためXレジスタをカウンタに用いて、図8―33のようにループを構成できますが、このループの構成にＵレジスタやＳレジスタは用いることはできないということです。これはうっかりしやすいので、念を押しておきます。Uレジスタはループのカウンタには使えません。

```
        LDX  #ループの回数…カウンタのセット
LOOP    ループ本体
          :
        LEAX  −1，X ………カウンタのデクリメント
        BNE  LOOP
          :
```

図8－33　Xレジスタによるループの構成

図8－34　6809 Mnemonics

ABX

Add B into X

SOURCE FORM：ABX

ADC

Add with carry into register

SOURCE FORM：ADCA (P)；ADCB (P)

ADD

Add into register

SOURCE FORM：ADDA　(P)；ADDB　(P)；
ADDD(P)

AND

Logical 'AND' into register

SOURCE FORM：ANDA (P)；ANDB (P)

ANDCC

Logical 'AND' immediate into CC

SOURCE FORM：ANDCC # dd

ASL

Arithmetic shift left

SOURCE FORM：ASLA；ASLB；ASL (Q)

ASR

Arithmetic shift right

SOURCE FORM：ASRA；ASRB；ASR (Q)

BCC, LBCC

Branch (short or long) if carry clear

SOURCE FORM：BCC dd；LBCC dddd

BCS, LBCS

Branch (short or long) if carry set

SOURCE FORM：BCS dd；LBCS dddd

BEQ, LBEQ

Branch (short or long) if equal

SOURCE FORM：BEQ dd；LBEQ dddd

BGE, LBGE

Branch (short or long) if greater than or equal

SOURCE FORM：BGE dd；LBGE dddd

BGT, LBGT

Branch (short or long) if greater than

SOURCE FORM：BGT dd；LBGT dddd

BHI, LBHI

Branch (short or long) if higher

SOURCE FORM：BHI dd；LBHI dddd

BHS, LBHS

Branch (short or long) if higher or same

SOURCE FORM：BHS dd；LBHS dddd

BIT

Bit test

SOURCE FORM：BITA (P)；BITB (P)

BLE, LBLE

Branch (short or long) if less than or equal to

SOURCE FORM：BLE dd；LBLE dddd

BLO, LBLO

Branch (short or long) if lower

SOURCE FORM：BLO dd；LBLO dddd

BLS, LBLS

Branch (short or long) if lower or same

SOURCE FORM：BLS dd；LBLS dddd

BLT, LBLT

Branch (short or long) if less than

SOURCE FORM：BLT dd；LBLT dddd

BMI, LBMI

Branch (short or long) if minus

SOURCE FORM：BMI dd；LBMI dddd

BNE, LBNE

Branch (short or long) if not equal

SOURCE FORM：BNE dd；LBNE dddd

BPL, LBPL

Branch (short or long) if plus

SOURCE FORM：BPL dd；LBPL dddd

BRA, LBRA

Branch (short or long) always

SOURCE FORM：BRA dd；LBRA dddd

BRN, LBRN

Branch (short or long) never

SOURCE FORM：BRN dd；LBRN dddd

BSR, LBSR

Branch (short or long) to subroutine

SOURCE FORM：BSR dd；LBSR dddd

BVC, LBVC

Branch (short or long) if overflow clear

SOURCE FORM：BVC dd；LBVC dddd

BVS, LBVS

Branch (short or long) if overflow set

SOURCE FORM：BVS dd；LBVS dddd

CLR

Clear

SOURCE FORM : CLRA ; CLRB ; CLM (Q)

CMP

Compare

SOURCE FORM : CMPA (P) ; CMPB (P) ;
CMPD (P) ; CMPX (P) ;
CMPY (P) ; CMPU (P) ;
CMPS (P)

COM

Complement (One's complement)

SOURCE FORM : COMA ; COMB ; COM (Q)

CWAI

Clear CC bits and wait for interrupt

SOURCE FORM : CWAI # dd

DAA

Decimal adjust accumulator A

SOURCE FORM : DAA

DEC

Decrement

SOURCE FORM : DECA, DECB, DEC (Q)

EOR

Exclusive 'OR'

SOURCE FORM : EORA (P) ; EORB (P)

EXG

Exchange registers

SOURCE FORM : EXG R1, R2

INC

Increment

SOURCE FORM : INCA, INCB, INC (Q)

JMP

Jump to address

SOURCE FORM : JMP dddd

JSR

Jump to subroutine at address

SOURCE FORM : JSR dddd

LD

Load register from memory

SOURCE FORM : LDA (P) ; LDB (P) ; LDD (P) ;
LDX (P) ; LDY (P) ; LDU (P) ;
LDS (P)

LEA

Load effective address

SOURCE FOMR : LEAX (T) ; LEAY (T) ; LEAU (T) ; LEAS (T)

LSL

Logical shift left

SOURCE FORM : LSLA ; LSLB ; LSL (Q)

LSR

Logical shift right

SOURCE FORM : LSRB ; LSRB ; LSR (Q)

MUL

Multiply accumulators

SOURCE FORM : MUL

NEG

Negate (Two's complement)

SOURCE FORM : NEGA ; NEGB ; NEG (Q)

NOP

No operation

SOURCE FORM : NOP

OR

Inclusive 'OR' into register

SOURCE FORM : ORA (P) ; ORB (P)

ORCC

Inclusive 'OR' immediate into CC

SOURCE FORM : ORCC # dd

PSHS

Push registers onto system stack

SOURCE FORM : PSHS (register list) : PSHS # dd

PSHU

Push registers onto user stack

SOURCE FORM : PSHU (register list) ; PSHU # dd

PULS

Pull registers from system stack

SOURCE FORM : PULS (register list) ; PULS # dd

PULU

Pull registers from user stack

SOURCE FORM : PULU (register list) ; PULU # dd

ROL

Rotate left

SOURCE FORM : ROLA ; ROLB ; ROL (Q)

ROR

Rotate right

SOURCE FORM : RORA ; RORB ; ROR (Q)

RTI

Return from interrupt
SOURCE FORM : RTI

RTS
Return from subroutine
SOURCE FORM : RTS

SBC
Subtract with borrow
SOURCE FORM : SBCA (P) ; SBCB (P) ;

SEX
Sign extend
SOURCE FORM : SEX

ST
Store register into memory
SOURCE FORM : STA (P) ; STB (P) ; STD (P) ;
STX (P) ; STY (P) ; STU (P) ;
STS (P)

SUB
Subtract from register
SOURCE FORM : SUBA (P) ; SUBB (P) ;
SUBD (P)

SWI
Sofware interrupt
SOURCE FORM : SWI

SWI2
Software interrupt 2
SOURCE FORM : SWI2

SWI3
Software interrupt 3
SOURCE FORM : SWI3

SYNC
Synchronize to interrupt
SOURCE FORM : SYNC

TFR
Transfer register to register
SOURCE FORM : TFR R1, R2

TST
Test
SOURCE FORM : TSTA : TSTB ; TST (Q)

(P)　イミディエイト，エクステンド，ダイレクト，もしくはインデックスアドレシングを含んだオペランド
(Q)　エクステンド，ダイレクト，もしくはインデックスアドレシングを含んだオペランド
(T)　インデックスアドレシングのみ含んだオペランド
R　レジスタのうちいずれかの指定
A, B, X, Y, U, S, PC, CC, DP, もしくはD
dd　8ビットデータ値
dddd　16ビットデータ値

基礎編の最後に

これで、基礎編の学習は全て終了です。基礎編では、マシン語の各命令を1つずつ解説し、簡単なプログラムを用いてその動作の確認を行ってきました。これによって、すべての（一部を除く）命令を一応扱えるようになったわけですが、まだ本格的なプログラム作りはしていません。そのあたりに不安を感じている読者もいるかもしれません。

そこで、この後の実践編では、基礎編で取りあえずわかる程度になったマシン語をさらに実践し、活用していく方法を学習します。そのため、この後の実践編では個々の命令ではなく、命令の組み合わせという点に重点を置き、さらに、FM—7シリーズ特有の事項の解説なども交えて、本格的なプログラム作りへ向っていくことになります。

もう既に峠は越しました。ここまで来れば後は比較的楽な道程と考えてけっこうです。いままでの学習を活かして、くじけずに学習していってください。

応　用　編

第 9 章

FM−7シリーズの構成と環境

——マシン語の観点から——

1. メモリマップ

基礎編の第3章で、FM—7シリーズのメモリマップについて記述しましたが、そのときにはマシン語をほとんど知らない状態でしたから、あまり詳しくは述べませんでした。そこで図9—1に詳しいメモリマップを示して、これについて解説していきます。(㋐、㋑は3章10節と同じ)

まず＄0000〜㋐までの領域は、BASICが、BASIC自身の作業用に使用している部分です。もし、この部分を破壊してしまうと、BASICが正常に動作しなくなったり、BASICインタプリタ自身が暴走するということにもなりかねません。ですから、私たちはこの部分にプログラムを格納したり、この部分を作業用に使用したりしてはいけないのです。しかし、逆の観点からみれば、この部分を上手に変換してやれば、BASICインタプリタを制御することもできます。例えばロードしたBASICプログラムがプロテクトされていた場合は、モニタや別に作成したマシン語プログラムなどによって、＄00D2番地に0を格納すれば、プロテクトを解除できます。

この部分に書き込んで変換するというのは非常に過激なことですから、十分にその意味を理解してから行わなければなりません（FM—7シリーズ解析マニュアルに詳細を記載しています)。反対に、ただ読み出して参照するのには一向にさしつかえありません。この部分には有用な情報が数多く格納されていますから、多いに利用するとよいでしょう。例えば、＄00C3番地には、画面の桁数（横の文字数）が格納されています。

㋐〜㋑が、フリーエリアで、BASICインタプリタが、私たちに使用することを許してくれた領域です。しかし、このフリーエリアは実はあくまでBASICプログラムのためのフリーエリアなのです。㋐からは通常BASICのプログラムが格納され、また㋑からアドレスの低い方へは文字変数が格納されます。これまで基礎編では、このことにお構いなく、＄5000番地や＄4000番地からプログラムを格納し、また＄6000番地からを作業領域として使用してきました。B

ＡＳＩＣのプログラムが全く格納されていなければ、このままでもよほどのことがない限り問題は生じません。ところが、長いプログラムが格納されていたり、文字変数をたくさん使用したりすると、マシン語のプログラムを破壊したりＢＡＳＩＣのプログラムを破壊したりして、最悪の場合、暴走ということにもなりかねません。

そこで対策は、ＢＡＳＩＣの“ＣＬＥＡＲ”命令の第２パラメータで行います。この第２パラメータは、文法書にもある様にＢＡＳＩＣの使用す

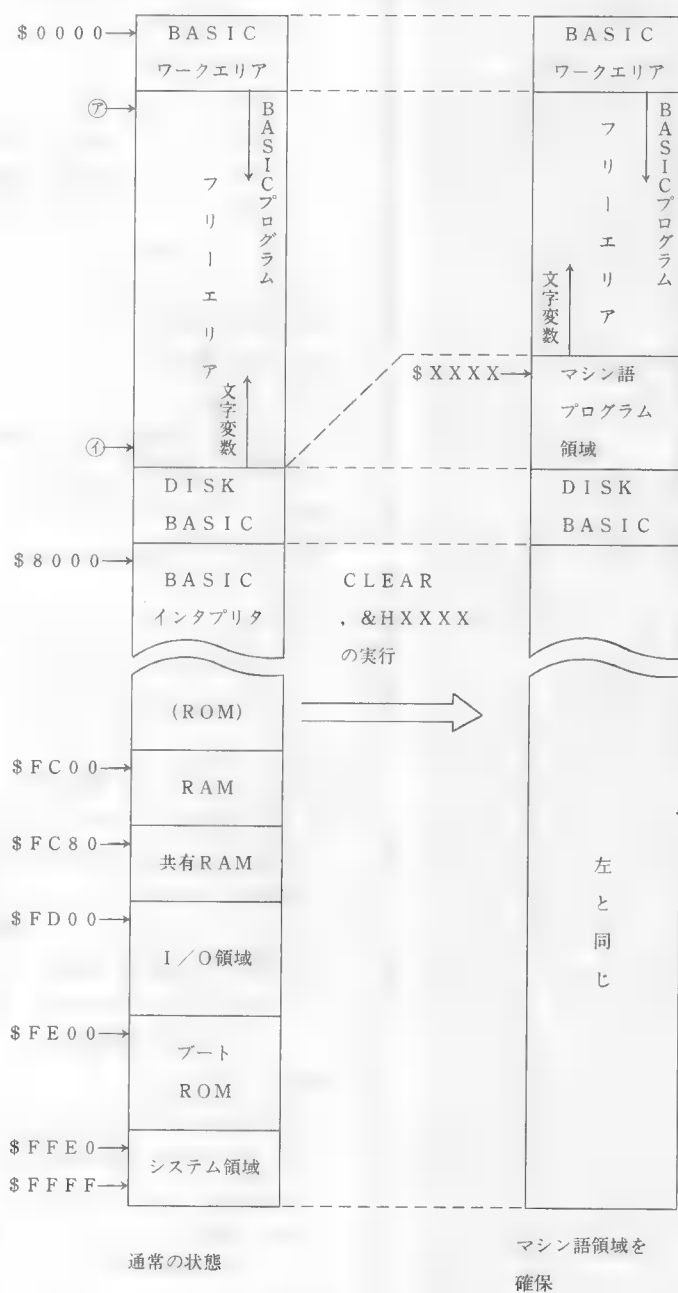



図９－１　メモリマップ（メインＣＰＵ）

る上限アドレスを設定します。例えば、

CLEAR,&H4FFF⏎

とすれば、BASICは$5000番地から②までを使用しなくなり、マシン語領域として開放してくれます。これによって開放された領域はBASICでは全く使用しないので、マシン語プログラムを格納したり、作業領域として自由に用いることができます。基礎編のプログラムも

CLEAR,&H4FFF⏎

を実行しておけば、BASICのプログラムを破壊したり、BASICによってマシン語プログラムが破壊されることはなくなり、BASICのプログラムと共存できます。

④から$8000までは、DISK BASICが格納されています。これは、DISK BASICで新たに使用できるBASICステートメントの処理を行う部分です。ですから、ROMモードでスタートした場合（ディスクがないときなど）には、この部分は存在せず、④＝$7FFFとなります。この部分は、BASICインタプリタの一部ですから、安易な書換えは、暴走やDISKの破壊につながりますから、十分注意してください（0～⑦までの領域以上に熟知している必要があります）。

$8000～$FBFFまでに、F－BASICのインタプリタがROMとして格納されています。この部分はROMですから書き換わる心配はありません。既に述べましたが（基礎編1章5節）、BASICのインタプリタは、各種のマシン語サブルーチンの集まりです。その中には、有用なものも数多くあるので、これを使用しない手はありません。例えば、Aレジスタに出力したい文字のアスキーコードを入れて、$D9D9をサブルーチンとして呼び出すと、画面にその文字が出力されます。この他にも数多くの有用なサブルーチンがありますので、解析マニュアルを参考にして試してみるのもよいでしょう（解析マニュアルには、各種サブルーチンがその使用例とともに紹介されています）。

しかし、これらのルーチンには、NEWをかけるルーチンなどのように、BASICプログラムを破壊するルーチンもあるので、十分その使い方を理解してから用いなければなりません。

$FC00～$FC7Fは、またRAMになっています。この部分はリセット時に（電源ONのときも）、一度使われる以外は、BASICは使用しないので、自由に用いることができます。ただし、大きさが小さいので、あまり用途は多くありません（後述する裏RAMを用いるプログラムではよくこの部分が使われます）。

$FC80～$FCFFは、FM－7の特徴的な部分です。詳しくは後に述べることにします。

$FD00～$FDFFは、I／O領域と呼ばれる部分です。この部分は、ディスクやプリンタ、ブザーなどの装置を制御する際に用います。この部分の各アドレスは各種の装置と関係しており、この部分に書き込んだり、読み込んだりすることによって、各種の装置を制御するわけです。例えば、$FD03番地に$41を書き込むと一定時間だけブザーが鳴ります。この部分の各アドレスの役割りについてここでは解説しませんが、必要なものについては、そのつど解説していきます。すべてのアドレスの詳細については、解析マニュアルかシステム仕様を参照してください。ところでこの部分以外では、書き込むことは危険を伴うこともありますが読み出すことは問題がありませんでした。けれども、このI／O領域に限っては、読み出すことにも危険が伴うことがあります。これは、指定されたアドレスを読み出すという動作だけによって制御する場合があるからです。

$FE00～$FFDFはブートROMです。この部分は本体後部のデップスイッチによって内容を3種に交換することができます。電源ONまたはリセット時の処理をする部分とディスクを読み書きするサブルーチンが含まれています。

$FFE0～$FFFFは、最後の2バイトがROMでそれ以外はRAMになっています。しかしこの部分もFM－7シリーズのシステムで使用しているので変換は禁物です。

以上がFM－7シリーズのメモリマップの概要です。いろいろ書きましたが、取りあえず、

CLEAR,&H××××⏎

とすれば、$××××＋1番地から②番地までが

マシン語プログラム用の領域として使用できるということを覚えておいてください。

2. サブCPUと裏RAM

FM—7シリーズの構成を考える場合、前節のメモリマップだけでは不十分です。というのはFM—7シリーズではサブCPUという非常に特徴的な構成を持っているからです。

FM—7シリーズのユーザーの方ならば、既にサブCPUの存在は十分承知のことと思います。このサブCPUは、キーボードの管理と画面表示などを担当しており、BASICインタプリタの実行などを担当しているメインCPUと二人三脚（2CPU3？）で処理を行います。これにより、1つのCPUですべての処理を担当するのに比べて、高速処理が期待できます（期待できるのですが、BASICインタプリタは必ずしもこの期待を満足しているとはいえません。だからこそ、マシン語のプログラムが望まれているとも言えます）。

このサブCPUという概念は非常に（いい意味で）特殊であると同時に複雑です。ですから、メインCPUとサブCPUとの連絡の取り方などを含めて後に1つの章を持って解説することにします。

FM—7シリーズの構成を考えるにあたって、もう一つだけ裏RAMについて解説します。前節メモリマップ（これはメインCPUのメモリマップでした。）の＄8000～＄FB00には、F-BASICのインタプリタがROMとしてどんと居座っています。もし、あなたがマシン語を理解し、大きなプログラムを組もうというとき、この大きなROMは、無用の長物となります。この事を考えて、この部分はROMとRAMを切り換えることができるようになっています。この切り換えた後のRAMはBASICは当然のことながら全く使用していないので、自由に使用することができます。

主な用途としては、マシン語プログラムにおけるデータ領域などがあります。この部分にマシン語プログラムを格納することも可能ですが、BASICのEXEC文などでは実行を開始させることができないこと、BASICインタプリタ内のサブルーチンを使用することができないなどの制限があります。

3. BIOSとROM内ルーチン

前節までは主にハードウェア的な面からFM—7シリーズの構成についてみてきました。次に、ソフトウェア的な面からFM—7シリーズでのマシン語プログラムの開発環境（開発が容易であるか）について考えてみます。

マシン語のプログラムの作成で機種ごとの違いが特に問題となるのは、I／Oルーチンです。I／Oルーチンというのは、マシン語レベルで、文字の出力や入力など、各種装置とのやりとりを行うルーチンのことです。CPUが同じであっても個々の機種の特徴から、I／Oルーチンの仕様やI／O領域の場所、各アドレスの意味などは全く違います。このため、同じ6809をCPUに持つ他の機種（例えば日立のS1など）にFM—7シリーズのプログラムを入力しても動きません。この対策として，大きなシステムプログラムなどでは、I／O（Input／Outputの略です。「入出力」という意味で用いられます）に関係する部分を、そうでない部分と分離して作成しています。こうすると、他の機種（CPUが違っては話になりませんが）に移植する場合でも、I／Oに関係する部分をその機種にあわせて書き換えるだけの労力しか必要ありません。

そこでFMシリーズ（7／8／11）では、このI／Oルーチンの仕様と使い方を同じにして、BIOSと呼ばれる一種のサブルーチンパッケージを装備しています。FM—7↔FM—8↔FM—11の間で、マシン語のプログラムであっても、比較的変更なしでプログラムを共有できるのは、このためです。さらにFM—7↔FM—NEW7↔FM—77間では、BIOSの仕様のみではなく、BASIC　ROM内のサブルーチンまで、ごく一部を除いて同じですから、これらの機種間では、

ＢＡＳＩＣはもちろんマシン語でもほとんどのプログラムが共有できます。

またＢＩＯＳは単に仕様を同じにして、プログラムを共有化できるだけではありません。ユーザがプログラムを作成する際にこのＢＩＯＳを使用してＩ／Ｏ処理を行えば、Ｉ／Ｏ装置の細かい制御に気を配る必要なしに、簡単に入出力を行うことができます。Ｉ／Ｏ装置の細かい制御は非常に繁雑で冗長なプログラムになってしまうので、ユーザがこのＢＩＯＳを使用することによってこの処理から開放されるのは、非常に喜ばしいことです。加えて、マニュアルにその使用法に関して公開されているという点も見逃せません。

さらに、既に述べたようにＦ－ＢＡＳＩＣのＲＯＭの内部には有用なサブルーチンが数多くつまっているので、これを利用するとプログラムを短くしたり、見通しをよくすること（プログラムのすじを追いやすくすること）ができます。このＲＯＭ内ルーチンに関しては、ＦＭ－７シリーズ付属のマニュアルには解説されていませんが、解析マニュアルには、その使用例とともに記載しました。

このように、ＦＭ－７シリーズのファームウェア（標準装備のソフトウェアのことをいう）は、マシン語プログラムの開発に（決して十分とは言えませんが）良い環境を提供してくれているといえるでしょう。

第 $ A 章

プログラミング

——アセンブラの使い方を中心に——

1. プログラミング言語

コンピュータとは、命令を与えなければ動作しませんから、人間はコンピュータに命令を与える方法を知らなければ、コンピュータを使うことはできません。コンピュータに命令を与えるためには、**プログラミング言語**というものを用いなければなりません。

既に私たちは、ＢＡＳＩＣ言語を知っており、また基礎編において、アセンブリ言語、マシン語を学習しました。世の中には、この他にも、図Ａ—1に示すように、数多くのプログラミング言語があります。図にあげましたが、プログラミング言語は大きく３つの部類に分けられます。

まず**インタプリタ**とは、ＢＡＳＩＣなどと同様に、人間が入力したプログラムを実行時にそのつ

	言　語
インタープリタ	BASIC LOGO LISP PROLOG FORTH APL　　　など
コンパイラ	FORTRAN RATFOR COBOL PASCAL C PL／I ADA ALGOL　　　など
	マシン語（機械語） アセンブリ語

図A－1　プログラミング言語

ど解釈して、インタプリタ内の適当なサブルーチンを呼び出し、命令を実行するものです。インタプリタは、実行するとき初めて解釈していく方法を取るため実行速度が遅いという欠点を持っています。しかし、会話型のプログラミングができ、暴走ということもめったにありませんから、初心者の学習や自分が考えたアルゴリズムの簡単な検証、たび重なるプログラムの変更（定数の変更など）を伴う場合などには重宝です。

コンパイラというのは、実行前にコンパイルという操作によって、人間の入力したプログラム（これは、ＢＡＳＩＣなどのように人間にわかりやすい文法で書くことができます）を全てそれと等価なマシン語プログラムに変換します。この操作によってプログラムは、マシン語のプログラムと同じ扱いをできるようになります。あとはマシン語のプログラムの実行と同じように実行できます。コンパイラを用いれば、一度コンパイルしてマシン語にしてしまえば毎度でもコンパイルなしで実行でき、また実行速度も速いという利点を持っています。一方、プログラムのごく一部の変更でも、再度コンパイルしなければならない（コンパイルには時間がかかることが多い）などの欠点も持っています。

そして、もう一つの分類が、**マシン語・アセンブリ言語**です。ところで、コンパイラを用いると、高級な（人間にわかりやすい）表現で書いたプログラムをマシン語に変換してくれるのですから、直接マシン語やアセンブリ言語で、人間が機械に譲歩して機械にわかりやすいかたちでプログラムを組むということは、不要に思われるかもしれません。しかし、コンパイラの作成してくれるマシン語は冗長になってしまうことが多く、直接プログラミングした場合に比べて、プログラムが長くなり、実行速度も落ちます。ですから、これだけ使い易いコンパイラなどが出現してもアセンブリ言語が残っているわけです。しかし、直接ハンドアセンブルによりマシン語を書き込む方法はさすがに少なく、アセンブラが用いられるのが普通となってきています。

2. *アセンブラの使い方*

さて、これまで基礎編で数多くのアセンブルリストをみてきましたが、これらはすべてＦＬＥＸと呼ばれるＤＯＳ（ＤＩＳＫ　ＯＰＥＲＡＴＩＮＧ　ＳＹＳＴＥＭの略）上で動作する、「ＴＳＣ－ＡＳＳＥＭＢＬＥＲ」（米国Technical Systems Consultants社製のアセンブラ）で作成されたものです。アセンブラには、この他にも富士通㈱の「**アブソリュートアセンブラ**」、ＯＳ９（これもＤＯＳ）上で動くマイクロウェア社製の「**インタラクティブアセンブラ**」など数多くあります。

これらのアセンブラは、それぞれの特徴を持つと同時に、アセンブラの起動方法、ソースプログラム（後述）の作成方法もそれぞれ違っています。それぞれについて使用法を解説するわけにはいきませんので、それぞれのアセンブラを購入したときに付属してくるマニュアルを参照していただきたいと思います。ここでは、FM—7シリーズ上で最もポピュラーと思われる富士通の「アブソリュ

〔図Ａ－２　プログラム〕

```
        ORG   $5000
        LDA   $6000,PCR
        BITA  #$08
        BEQ   OFF
        LDA   #$FF
        BRA   STORE
OFF     CLRA
STORE   STA   $6001,PCR
        JMP   $ABF4
        END
```


〔図Ａ－３　入力例１〕

```
100 '        ORG $5000
110 '        LDA $6000,PCR
120 '        BITA #$08
130 '        BEQ OFF
140 '        LDA #$FF
150 '        BRA STORE
160 'OFF     CLRA
170 'STORE   STA $6001,PCR
180 '        JMP $ABF4
190 '        END
```

ートアセンブラ」を例に取って、その使用法の概略を解説します。(本書では、解説の都合上この節を除いてすべてＴＳＣ社のアセンブラを用いていますが、そのまま富士通のアブソリュートアセンブラでも動作できるように配慮してあります。富士通のものとの違いは、アセンブルリストの形式だけにおさえてあります)。

例として、図Ａ—2のプログラムを考えます。

まず、ＢＡＳＩＣで図Ａ—3のようにプログラムを入力します。ここで注意しなければいけないのは、入力したプログラムは当然のことながらＢＡＳＩＣのプログラムではないので“'”（シングルコーテーション）をつけて、注釈文の形で入力します。行番号はどう付けてもかまいません（本来、特にＤＯＳ上のアセンブラでは、プログラムの入力を「エディタ」と呼ばれる編集プログラムで行うのですが、ＦＭ—7シリーズでＤＯＳなどを用いない場合には適当なエディタがないので、ＢＡＳＩＣで代用するわけです)。この入力の際、連続するスペースは１文字のスペースに省略できますので、図Ａ—4のように入力しても構いません。

次に、このプログラムをディスクにセーブします。セーブは常にアスキー形式で行わなければいけませんから、

〔図Ａ－４　入力例２〕

```
100 ' ORG $5000
110 ' LDA $6000,PCR
120 ' BITA #$08
130 ' BEQ OFF
140 ' LDA #$FF
150 ' BRA STORE
160 'OFF CLRA
170 'STORE STA $6001,PCR
180 ' JMP $ABF4
190 ' END
```

〔図Ａ－５　アセンブルのしかた〕

```
LIST                                                  ……注釈文にする（ＲＥＭは不可）

100 '        ORG   $5000
110 '        LDA   $6000,PCR
120 '        BITA  #$08
130 '        BEQ   OFF
140 '        LDA   #$FF           ……ソースプログラム（この様に入力する．図Ａ－４のように入力
150 '        BRA   STORE                                してもよい）
160 'OFF     CLRA
170 'STORE   STA   $6001,PCR
180 '        JMP   $ABF4
190 '        END

Ready
SAVE "SAMPLE",A ……アスキー形式でセーブする

Ready
■

RUN "ASM09" ……アセンブラの起動（アセンブラはドライブ０にいれておくこと）
FM-7 MBL6809 ASSEMBLER V1.1 BEGIN. ……起動したことを示すメッセージ

SOURCE FILE DESCRIPTOR ? ="SAMPLE"
SOURCE FILE WAS OPENED.          ……ソースプログラムのファイルディスクリプタを入力する
LIST DEVICE ? ="LPT0:" ……アセンブルリストを出力する機器を指定する（ここではプリンタ）
LINE PRINTER WAS OPENED.
OBJECT FILE DESCRIPTOR ? ="SAMPLE.O"
OBJECT FILE WAS OPENED.          ……オブジェクトプログラムを出力するファイルディスクリプ
                                   タを入力する
TOTAL ERRORS 00000--00000
TOTAL WARNINGS 00000--00000      ……エラーと警告のレポート
                                 ……エラーなし
PROGRAM BEGIN ADDR=5000
PROGRAM END   ADDR=5013          ……プログラムのアドレスを出力してくれる
PROGRAM ENTRY ADDR=**** ……プログラムの実行開始アドレスを出力してくれる
ASSEMBLER END.      （ここでは，ソースリスト中に指定がないので＊＊＊＊となっている）

Ready
```

SAVE "ファイル名", A⏎
とします。これは全く同じ動作をする、
LIST "ファイル名" ⏎
でも構いません。また、ディスクでなくカセットでも手間さえ厭わなければ可能です。詳しくはマニュアルを参照してください。ここではディスクとして解説します。

これでディスク上にプログラムが格納されました。この状態ではまだアセンブリ言語のまま（これを**ソースプログラム**または**ソースファイル**と呼びます）ですので、次にアセンブラを用いてアセンブルします。図A―5に示すように、アセンブラを起動して、いくつかの質問に答えます。すると（この場合にはプリンタ。画面を指定すれば画面に）、**アセンブルリスト**が出力されます。(図A―6)ここでエラーがあったときには、初めに戻ってソースプログラムを修正します。エラーがなければ、ディスク上に指定したファイル名で、**オブジェクトプログラム**（できあがったマシン語プログラムのこと。オブジェクトファイルともいう）ができあがっています。図A―7は、それをロードして確認した例です。

以上の手順で、アセンブルが完了します。

〔図A－6 出力されたアセンブルリスト〕

〔図A－7 ロードしてみる〕

```
LOADM "SAMPLE.O"··········オブジェクトプログラム（マシン語プログラム）のロード

Ready
MON··························モニタへ

*D5000·······················$5000番地からダンプ

5000 A6 8D 0F FC 85 08 27 04 ┐
5008 86 FF 20 01 4F A7 8D 0F ├···たしかにプログラムがある
5010 F0 7E AB F4 00 00 00 00 ┘
5018 00 00 00 00 00 00 00 00
5020 00 00 00 00 00 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*
```

◀参考：ＴＳＣ社アセンブラの使い方▶

富士通のアセンブラは、ＤＯＳを使用しないので、Ｆ－ＢＡＳＩＣの制御下で動作させることができます。これに対して、ＦＬＥＸやＯＳ９などのＤＯＳ上で動作するアセンブラの場合、その使用法に若干違いがあるので、ＦＬＥＸ上で動作する、ＴＳＣ社のアセンブラを例に取って使い方を紹介しておきます。

まずアセンブリ言語のソースプログラムを、エディタを用いて作成します。このエディタは、独立した編集用プログラムです。ＢＡＳＩＣの場合プログラムの編集と実行は同じＢＡＳＩＣ上で行うことができましたが、アセンブラやコンパイラなどでは、言語内に編集機能は持っておらず、ソースプログラムの入力は全てエディタによって行わなければいけません。このようにエディタは編集専用ですから、その編集機能はＢＡＳＩＣのそれよりも格段と向上しており、文字列のサーチや置き換えなどもサポートしています。もしあなたが長いプログラムを作成するのであれば、これらの機能は不可欠です。ＦＬＥＸでは、作成するプログラムの名前をＳＡＭＰＬＥとすると、プロンプト（入力を促がす記号）の“＋＋＋”に続いて、

＋＋＋EDIT SAMPLE⏎

とすることによってエディタが起動します。各種のエディタコマンド（マニュアルを参照）を用いて、ソースプログラムを入力し終えたら、

＃S ⏎

でエディタを抜けます。これにより、ＤＩＳＫ上にＳＡＭＰＬＥ．ＴＸＴというファイルが作成されます。

次に、ソースプログラムを

＋＋＋ＡＳＭＢ　ＳＡＭＰＬＥ⏎

でアセンブルします。エラーがなければ、ＤＩＳＫ上に、ＳＡＭＰＬＥ．ＢＩＮというファイルが作成されます。これがオブジェクトプログラムです。これを実行するには、

＋＋＋ＳＡＭＰＬＥ．ＢＩＮ⏎

と入力します(図Ａ—8にアセンブルの手順を例示しました)。

〔図Ａ－８　ＴＳＣ社のアセンブラによるアセンブルの例〕

```
+++EDIT SAMPLE ◀………… “SAMPLE”というソースプログラムを作成する
NEW FILE◀…………………… 新しいファイルであることを示す
    1.00= ORG $5000          ┐
    2.00= LDA $6000,PCR      │
    3.00= BITA #$08          │
    4.00= BEQ OFF            │
    5.00= LDA #$FF           │
    6.00= BRA STORE          ├………… プログラムを入力する
    7.00=OFF CLRA            │
    8.00=STORE STA $6001,PCR │
    9.00= RTS                │
   10.00= END                ┘
   11.00=# ◀…………………………… 入力を終了する
   10.00= END
#6C/0/O/ ◀………………………… 6行目の0（ゼロ）をO（オー）に直す
    6.00= BRA STORE …………… 直っている
#^P! ◀……………………………… 最初から最後まで表示する
    1.00= ORG $5000          ┐
    2.00= LDA $6000,PCR      │
    3.00= BITA #$08          │
    4.00= BEQ OFF            │
    5.00= LDA #$FF           │
    6.00= BRA STORE          ├………… 表示されたプログラム
    7.00=OFF CLRA            │
    8.00=STORE STA $6001,PCR │
    9.00= RTS                │
   10.00= END                ┘
```

```
#S◄·······················································エディタをぬける
+++ASMB SAMPLE◄·······································"SAMPLE"をアセンブルする
  5000                          ORG     $5000
  5000 A6    8D OFFC            LDA     $6000,PCR
  5004 85    08                 BITA    #$08
  5006 27    04                 BEQ     OFF
  5008 86    FF                 LDA     #$FF
  500A 20    01                 BRA     STORE
  500C 4F             OFF       CLRA
  500D A7    8D OFF0  STORE     STA     $6001,PCR
  5011 39                       RTS                  ············アセンブルリスト
                                END

O ERROR(S) DETECTED

SYMBOL TABLE:

OFF     500C    STORE   500D

+++CAT 0 SAMPLE◄·····················ドライブ0の"SAMPLE"というファイルを出力する

CATALOG OF DRIVE NUMBER 0
DISK: FLEX   #1

 NAME    TYPE   SIZE   PRT

SAMPLE   .TXT     1◄··········································ソースプログラム
SAMPLE   .BIN     1◄··········································オブジェクトプログラム

SECTORS LEFT = 837

+++
```

3. アセンブラ擬似命令

基礎編のアセンブルリストには、必ず最初に

ORG $××××

そして、最後に

END

という命令が加えられていました。これらの命令は既にみているように、マシン語の命令ではなく、アセンブラに対して指示を与える命令で、CPUの動作には直接関係しないので、アセンブラ擬似命令と呼ばれます。この擬似命令は使用するアセンブラにより少しずつ異なっていますが、ここでは、少くとも富士通とTSC社のアセンブラには共通である部分について解説します（本書ではここに述べる以外の擬似命令は使用していません）。

◀ORG（Origin）命令▶

ORG 命令は、この命令以降のプログラムを格納する番地を指定する命令です。この命令には、通常ラベルをつけることは禁止されています。もしプログラムを＄５０００番地から格納したいのであれば

ORG $5000

とすればよいわけです。このORG 命令はプログラム中にいくつ置いても構わないので、途中でORG 命令を置くことにより、プログラムを分割することもできます。

◀END（End of Source Program）命令▶

この命令は、アセンブラにソースプログラムの終りを指示します。ソースプログラムの最後には必ずこの命令を置かなければいけません。この命令にラベルをつけることはできませんが、

END $5000

のようにオペランドをつけることができます（省略可）。このオペランドの値は、オブジェクトプログラムの実行開始アドレスとして使用されます。前節の図A—6では実行開始アドレスが“＊＊＊＊”

となっていましたが、END命令にオペランドをつけて実行開始アドレスを明示すると、きちんとアドレスが表示されます。

しかし、このオペランドに数値を直接書くことは少く、実行開始したい場所にラベルをつけて、

```
        START LDA  #$00
                   ⋮
                END START
```

などというように用いるのが普通です。

◀ＥＱＵ（Equate）命令▶

これまで用いてきたように、ラベルはアドレスの数値のかわりに文字列を用いたものでした。アセンブラでは、アドレスのかわりだけでなく定数などのかわりに、文字列を用いることもできます。ラベルも含めて、数値を文字列で置き換えたものを**シンボル**といいます。このシンボルは、6809のアセンブラの場合、通常**英文字で始まる６文字以内の英数字**となっています。

EQU 命令は、シンボルに値を定義する命令で

```
  ABC   EQU   123
```

と用います。この例では、“ＡＢＣ”というシンボルに123という値が定義されます。こうしておいて、プログラム中の他の所で、例えば、

```
  LDX #ABC
```

としたとすれば、これは、

```
  LDX #123
```

と全く同じとみなされるわけです。

しかし、この例で、

```
  LDX #$ABC
```

としてしまうと、16進の＄ＡＢＣとみなされてしまいます。つまり、シンボルというのは、文字列の置き換えではなく、値の置き換えであるので、基数を示す＄や％などをつけてはいけません。

この EQU命令でよく用いられる表現で、

```
  ADR  EQU  *
```

というのがあります。オペランド部にある＊は、常に、命令が格納されるアドレスを示している変数です。ですから、この例では、この行があるアドレスにラベルをつけるのと同じことです。もう一つ例をあげれば

```
  BRA  *
```

というのは、自分自身に分岐する命令となるので無限ループとなります。

ところでこれまで、私たちはプログラムの中で使用する作業領域（結果などを入れておく場所）として、主に＄６０００番地以降を使用し、さらに、その作業領域をオペランドで指定する場合には、

```
  STA  $6000, PCR
```

と、アドレスを全て数値で与えてきました。短いプログラムなどでは＄６０００番地の用途もすぐにわかるかもしれませんが、長いプログラムではそうはいきません。そこで、ここにもシンボルを用います。例えば＄６０００番地を結果の格納場所として用いるのであれば、プログラムの先頭付近で

```
  KEKKA EQU  $6000
```

として“ＫＥＫＫＡ”というシンボルを定義し、この番地を使用するときには

```
  STA KEKKA, PCR
```

と用いると非常にわかりやすくなると同時に間違いもおこりにくくなります。

以上から、**シンボルを積極的に用いるのはきれいな、わかりやすい、間違いのないプログラムを作るのに必要不可欠な事項**といえるでしょう。

◀ＦＣＢ（Form Constant Byte）命令▶

マシン語のプログラムを３つの部分に分けるとすると、**実行部**（ＣＰＵによって実行される部分）、**定数部**（実行に必要なデータのある部分）、**作業領域**（実行結果や途中結果を保持しておく部分）に分けられます。アセンブリ言語で実行部をどのように記述したらよいのかは既におわかりのことと思います。定数部を記述するのが、FCB FDB、FCC の擬似命令です。

FCB 命令は、１バイト（つまり８ビット）の定数を記述する命令です。例えば

```
  TEISU FCB 123
```

とすると、123という１バイトの定数がプログラム中におかれて、そのアドレスに“ＴＥＩＳＵ”というラベルがつきます。

前の EQU 命令と混同してしまう人も多いので詳しく解説しましょう。

EQU 命令と FCB 命令との大きな違いは、オペランドに書いた数がプログラム中に置かれる（“展開する” ともいいます）かどうかの違いです。

CONST EQU 123

とした場合には、ＣＯＮＳＴというシンボルは、123だとアセンブラは理解しますが123という数値はプログラム中には置かれません。別の１箇所で、

LDA #CONST

などと用いる場合は別ですが、プログラム中に他の所でＣＯＮＳＴというシンボルを用いなければプログラム中には全く表れてきません。

これに対して、

TEISU FCB 123

とした場合には、アセンブラはプログラム中に(たとえ他の所でＴＥＩＳＵというシンボルが使われようと使われまいと)123という１バイトを置きます。さらに、この場合、ＴＥＩＳＵというシンボルは、123を表すシンボルではありません(FCB 命令ではシンボルをつける必然性はありません)。ＴＥＩＳＵは、123という数が格納してあるアドレスを示すラベルですから、もし、この命令が＄５４３２番地からアセンブルされたとすれば、ＴＥＩＳＵは＄５４３２を表すシンボルとなります。

簡単にいえば、 EQU はアセンブラに単に文字列を数値に置き換えてくれることを望むときに用いるものですが、 FCB はアセンブラにプログラム中に入れ物を確保して、そこに定数を入れて置くようにする命令です。

◀ＦＤＢ (Form Double Byte) 命令▶

これは、 FCB 命令が１バイトの定数を記述する命令であったのに対して、２バイトの定数を記述する命令です。再度注意しておきますが、

CPU FDB 6809

とした場合、ＣＰＵというシンボルは6809という数ではなく、6809という定数の格納されているアドレス、FDB では２バイトになるので、上位８ビットの格納されているアドレスを示すラベルです。

FCB 命令と FDB 命令では、オペランドをコンマ（，）で区切ってつなげることができます。例えば

FDB 6809
FDB 6502

とするかわりに

FDB 6809, 6502

とすることができます。ただし（当然のことながら） FCB と FDB を混同して続けることはできません。もし

FDB $ABCD
FCB $EF

のつもりで、

FDB $ABCD, $EF

とすると、メモリ上には、

＄ＡＢ，＄ＣＤ，＄００，＄ＥＦ

というように、＄ＥＦが２バイトに拡張されて、格納されてしまいます。

◀ＦＣＣ (Form Constant Character) 命令▶

プログラムの定数部に、画面に出力するメッセージなどを格納して置きたいことがよくあります。この場合、例えば“ＡＢＣ”という文字列だったとすれば、１文字ずつアスキーコードに変換して

FCB $41, $42, $43

としても構わないのですが、非能率的です。このときの＄４１は“Ａ”のアスキーコードです。

こういった要求に応えてくれるのが FCC 命令です。前の例の場合、

FCC “ABC”

とします。格納したい文字または文字列を、文字列中で用いない適当な記号（数字以外の文字でもよいが、文字列本体と間違いやすいので、通常は“／”、“〃”、“’”、“＄”などを用います）で両側からはさんで記述します。

FCC 命令にもラベルをつけることができますが、これも FCB 命令などと同様、アドレスを示すラベルですから注意してください。

◀ＲＭＢ (Reserve Memory Bytes) 命令▶

プログラムの実行部と定数部の記述法は以上のようになっています。残った作業領域の記述にこのRMB 命令を用います。

前に EQU 命令を用いて作業領域を指定する方法を示しました。しかし EQU 命令による指定を

用いると、連続した作業領域の指定の場合、図A—9となり繁雑となります。また、EQU命令による場合には、作業領域の確保が明らかでないという欠点があります。

そこで、作業領域をプログラム中で確保する場合を図A—10にあげました。RMB命令はオペランドに書かれた数のバイト数を作業領域として確保します。この命令を用いるとアセンブルの際アドレスをオペランドのバイト数だけ増やして領域が確保されるのですが、この領域にどんな数が設定されるかは決まっていないので注意する必要があります（これはEQU命令を用いた場合も同じです)。

これによって、EQU命令ではなくRMB命令を用いると、各領域が何バイト使用するかが明確にわかります（EQU命令では“KEKKA 3”のバイト数がわからない)。

4. アルゴリズムからフローチャートへ

アセンブラの使い方が、ひととおりわかったところで、プログラムを１本実際に組んでみることにしましょう。

例として取りあげるのは、有名な８(エイト)クイーン問題です。プログラムは、８クイーンの全ての解を表示するプログラムとします。８クイーン問題というのは図A—11の８×８のチェス盤上に８つのクイーン（縦、横、斜めにいくつでも動ける）を、それぞれがお互いに取れないような位置に配置するという問題です（図A—11はこの１つの解です)。

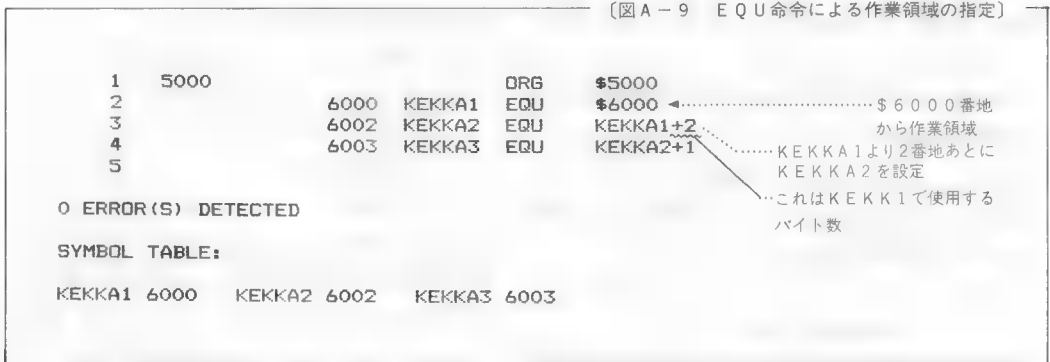
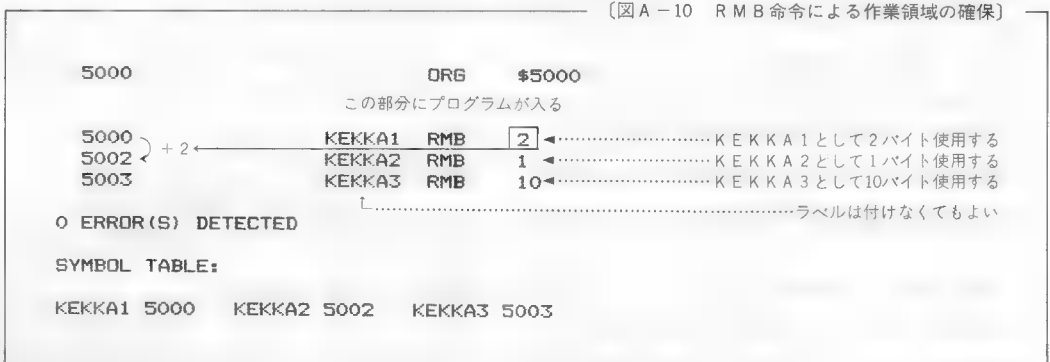
まずは、アルゴリズムを考えなければなりません。これはパズル的にも非常におもしろいので、少しの間各自で必ず考えてみてください。

ここで採用するアルゴリズムはもっとも単純な総当りによる方法です。つまり８つのクイーンをとにかく配置して、条件を満たしているかどうか

〔図A－9　EQU命令による作業領域の指定〕

```
1    5000                         ORG    $5000
2                6000   KEKKA1    EQU    $6000
3                6002   KEKKA2    EQU    KEKKA1+2
4                6003   KEKKA3    EQU    KEKKA2+1
5

O ERROR(S) DETECTED

SYMBOL TABLE:

KEKKA1 6000    KEKKA2 6002    KEKKA3 6003
```


〔図A－10　RMB命令による作業領域の確保〕

```
5000                    ORG    $5000
              この部分にプログラムが入る
5000 ⌒ +2 ←  KEKKA1   RMB    2  ←……ＫＥＫＫＡ１として２バイト使用する
5002 ⌒        KEKKA2   RMB    1  ←……ＫＥＫＫＡ２として１バイト使用する
5003          KEKKA3   RMB    10 ←……ＫＥＫＫＡ３として10バイト使用する
              ↑……………………………………………………ラベルは付けなくてもよい
O ERROR(S) DETECTED

SYMBOL TABLE:

KEKKA1 5000    KEKKA2 5002    KEKKA3 5003
```

をチェックし、満たしていなければ配置を少し変えて再度トライするという方法です。しかし、配置換えは規則的に行わなければ、全ての場合をくまなく検査できません。

そこで、解となる配置では、各列に1つのクイーンしか存在できないことを利用して、次のアルゴリズムを使用します。基本的な部分は、

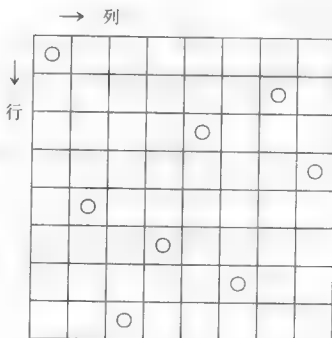
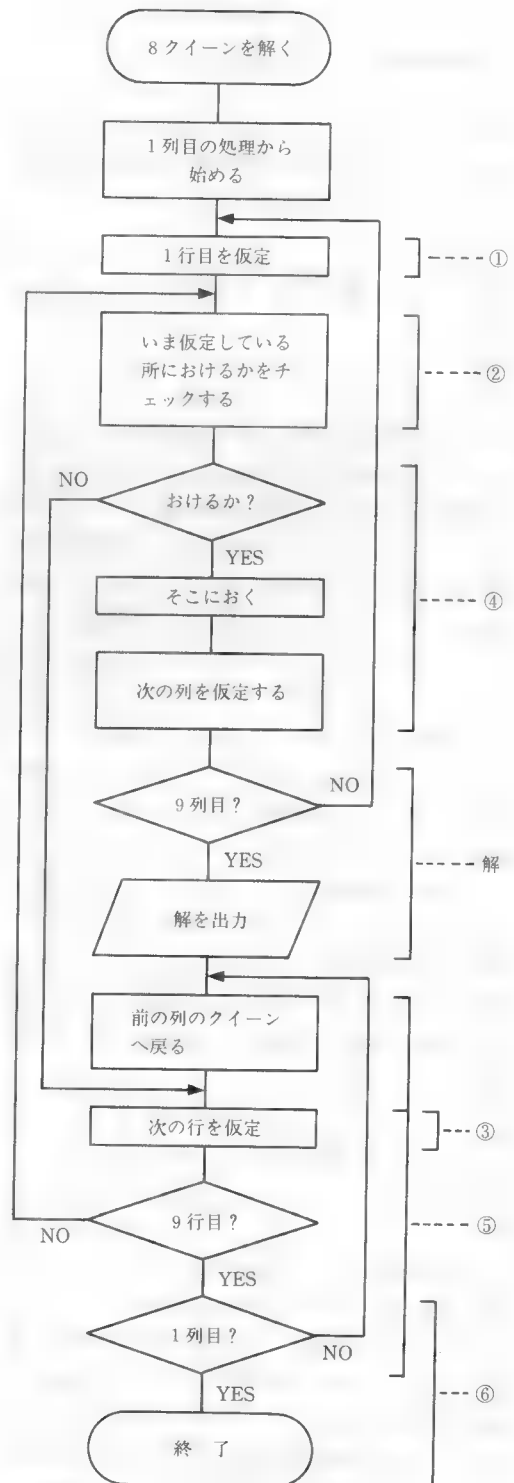
①i－1列目まで配置ができたら、i列目の1行目にクイーンを置いてみる。
②条件を満足しているかをチェックする。
③満足していないときは、クイーンを1行下げて②へ
④満足しているときは、次の列（i＋1列目）の処理へ移る。

となります。この手順で、8列めのクイーンまできちんと配置できたときに、それが1つの解となります。次に細い箇所では③で1行下げられない場合の処理です。

⑤1行下げられない場合には、前の列のクイーンを1行下げて処理を続ける。
⑥さらに、1列目で1行下げられない場合には、全ての場合を検討し終ったので終了する。

以上がアルゴリズムです（より良いアルゴリズムはまだまだたくさんありますが、取りあえずは、このアルゴリズムを採用します)。

つまり、各列のクイーンの場所を何行めかで示すとすれば、原理的には、

1 1 1 1 1 1 1 1→1 1 1 1 1 1 1 2→
1 1 1 1 1 1 1 3→ ……
…… →1 1 1 1 1 1 1 8→

図A－11　8クイーン

図A－12　フローチャート

1 1 1 1 1 1 2 1→ ……

…… →8 8 8 8 8 8 8 8

この順で検討していくわけです。

このアルゴリズムをフローチャートにしたのが図A—12です。アルゴリズムどおりであるかを各自点検してください。ここまででプログラムの実行部の構造が明らかになりました。

5. データ構造をどうするか

フローチャートができると次は、データ構造を考えなければなりません。データ構造というのはプログラムの処理に必要な情報（この場合には盤上の配置）を数でどう表現するかということです。

8クイーンのプログラムの場合、一番問題となるのは、盤をどのように表現するかです。1つの方法は、8×8の配列のようなものを用意して、クイーンがあれば1、なければ0とする方法です。しかし、この方法では、フローチャートを作成した時に考慮した、「解となる配置では各列に1つのクイーンしか存在できない」という点を全く無視してしまうことになります。

そこで、ここでは盤を1×8の配列とすることにし、各々の配列には、その列のクイーン位置を0～7で表わすことにします。具体的にいえば、図A—11の状態を0 4 7 5 2 6 1 3と8つの数字で示そうというわけです。こうすれば先の方法に比べて使用するバイト数は8分の1になり実行速度もアップします。

6. マシン語対応フローチャート

処理の概要とデータ構造が決まれば、後はいわゆるプログラミングになるわけです。

まず、データ構造のマシン語での実現方法を考えます。この場合、盤をどういう方法で実現するかが問題となります。データ構造の項で述べましたが、1×8の配列を使用します。配列といえば定石としてインデックスアドレッシングを使用します。この場合は配列の添字にあたるもの（何列めかを示す数）が動的に変化するので（次の列に移ったり、前の列に戻ったりします）添字としてアキュームレータA、インデックスレジスタとしてXレジスタを採用すれば、この配列を

LDB A,X

と扱うことができます。このレジスタの割当ては重要で、ここでうまくやらないとプログラムが冗長になったり、実行速度が落ちたりします。

ここまでくればプログラムは50%完成したと思ってさしつかえありません。あとは前に作成したフローチャートにそってマシン語の命令を組み合わせていけば良いわけです。この時点で一気にコンピュータに向ってプログラムを組んでも構わないのですが、それではあまり教育的とはいえませんのでまず、**マシン語対応フローチャート**を作成します。このマシン語対応フローチャートというのは図A—12のフローチャートで文章で書いていた処理を、マシン語の命令に対応した形で詳しく記述するものです。図A—13がそのマシン語フローチャートです。

図A—13①の部分については図A—12と同じですから理解できるでしょう。このマシン語対応フローチャートの表記法は全くのパーソナルなものですからこれでなければいけないというわけではありません。各自がわかりやすく書いて構いません（私の表記法で例えば

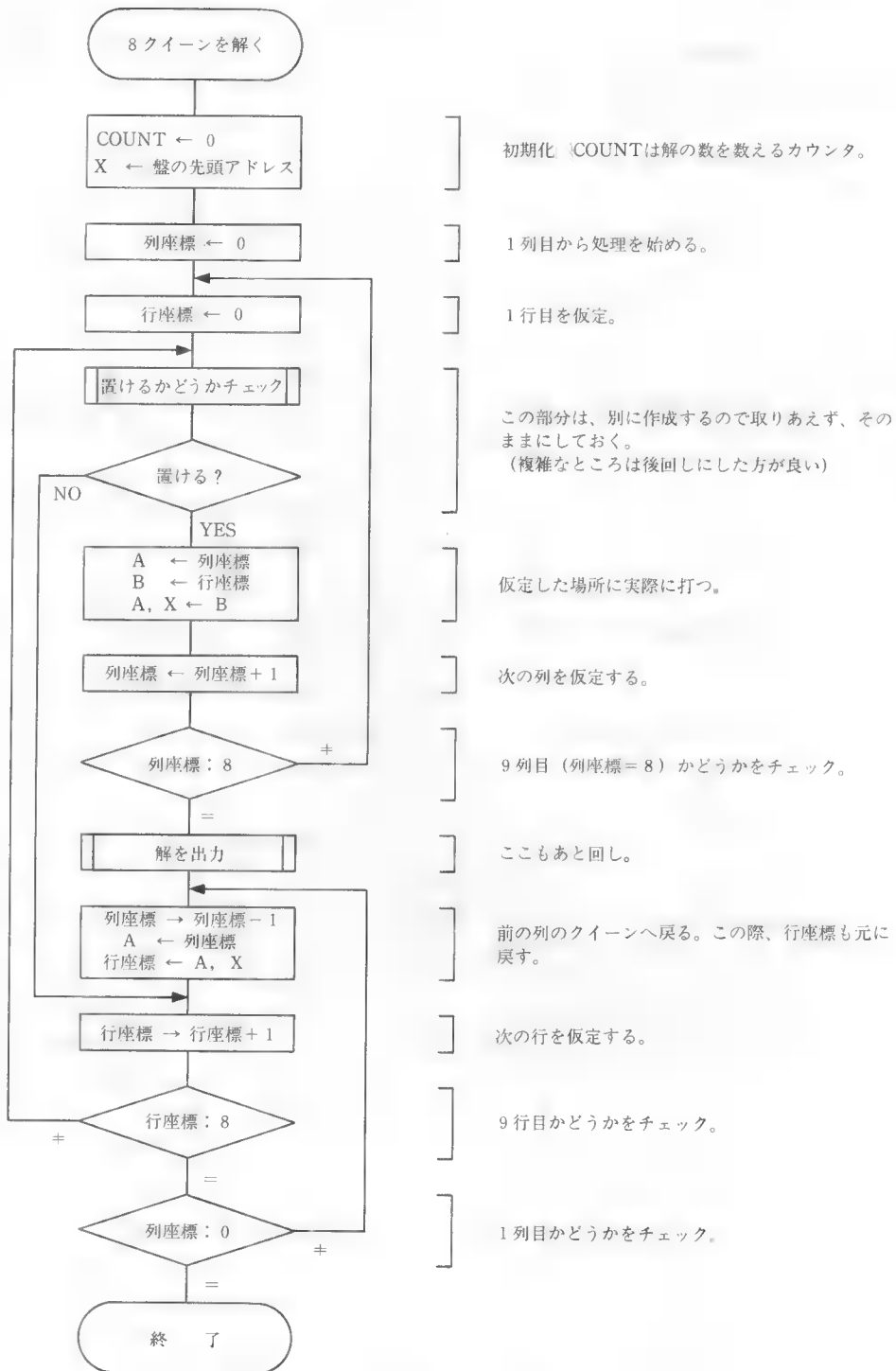
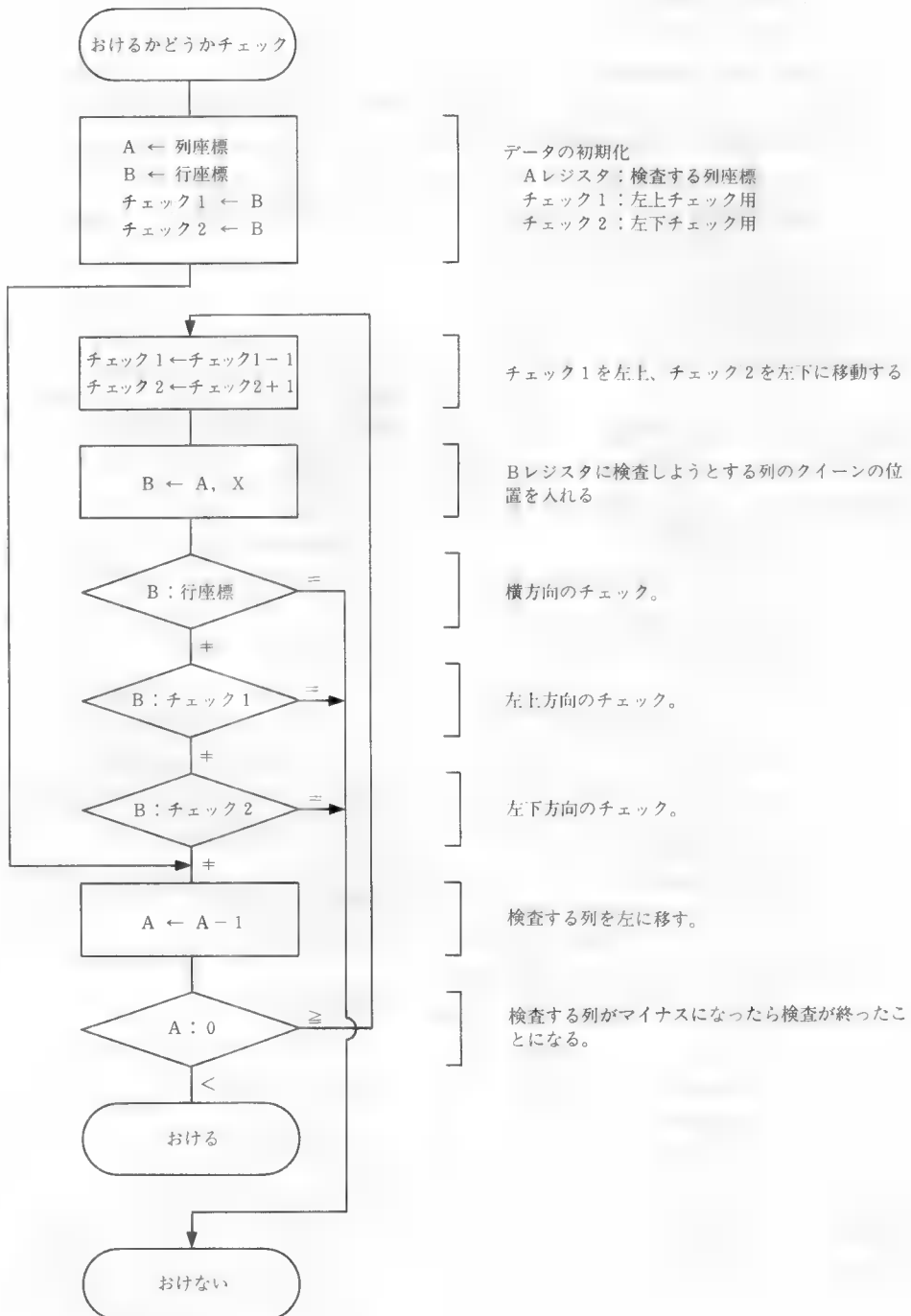
A← 列座標

は

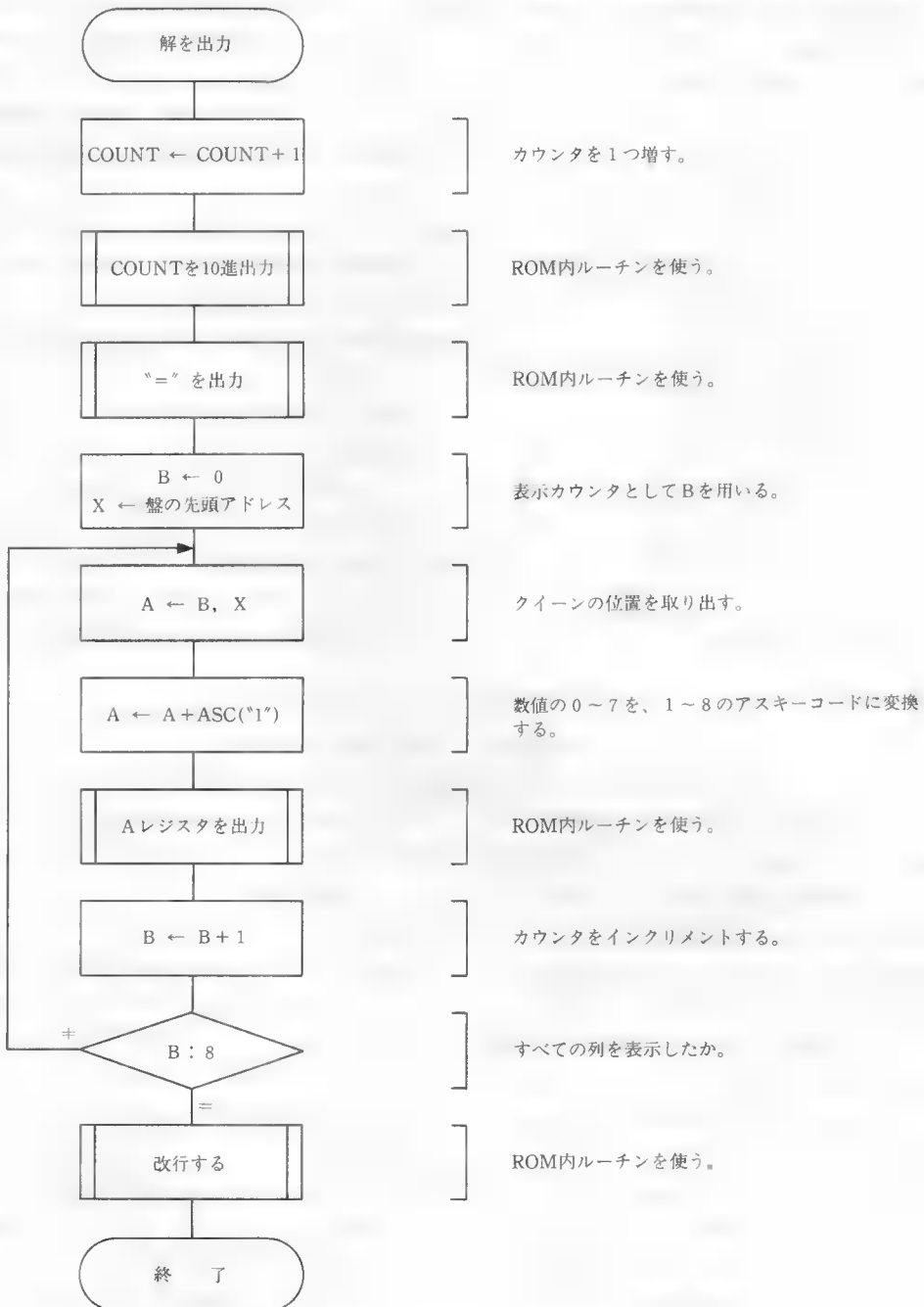
LDA 列座標

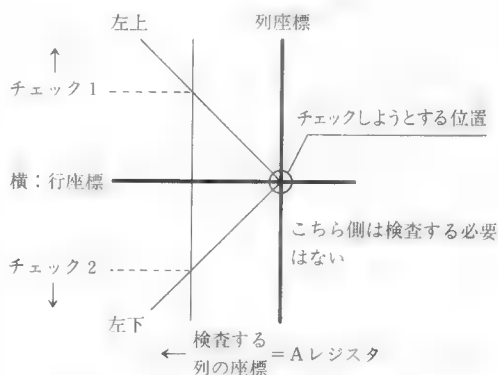
の意味です)。また、細かいところは後回しにしていくと、書きやすくなります。

図A—13②は，おけるかどうかをチェックする部分のフローチャートです。このルーチンでは、Aレジスタに検査する列の座標を入れています。チェックしようとする位置より右側はまだクイーンは置いていませんから検査する必要はありません。プログラムは、検査する列のクイーンの位置が、あってはならない位置（左上、左下、横）にないかどうかをチェックする方法を取っています（図A—14参照)。

図A—13③は、解を出力する部分のフローチャートです。表示は、

図A－13①　マシン語対応フローチャート(1)

図A－13② マシン語対応フローチャート(2)

1＝15863724 (図A—11の場合) というようにすることにします。"＝"の前には何番目の解であるかを表示することにします。

もうここまでくれば80%完成です。後は、コンピュータにアセンブリ言語で入力すれば良いわけです。もし、マシン語対応フローチャートをみながら、すぐにキーボードから打ち込むということ

図A—13③ マシン語対応フローチャート(3)

ができない（初めのうちは当然ですが）のであれば、一度コーデングという過程を経て実行してください。コーデングというのは、マシン語対応フローチャートをみながら、アセンブリ言語でプログラムを書くことです。つまり、図A—15の右側のソースプログラムの部分を紙に書いてみるわけです。慣れてくれば、コーデングは省いても構わないでしょう。

図A—14　おけるかどうかのチェック

7. プログラムの完成

図A—15が完成したプログラムをアセンブルして得られたアセンブルリストです。このプログラムでは、1ヶ所書きかえる（＄5002番地）だけでnクイーンが求められるようにしてみました（n≧1、nが10以上だと表示がおかしくなりますが動作はします)。ここでは最左に表示されている行番号を参照してアセンブラ擬似命令を中心に解説します。

1〜7行：この部分は注釈です（BASICのREM文と同じ)。行の最初に（スペースを入れずに）“＊”を置くとその行は注釈となりアセンブラは無視してくれます。

8行：プログラムは＄5000番地から格納します。この行のオペランド（オペランドがないときにはニーモニック）のあとに1つ以上のスペースを入れれば、その後は注釈文となります。

10行：NQUEEN（クイーンの数）を8にします。

13行：NQUEENをメモリに格納して、そのアドレスにDIMのラベルをつけます。こうすると、この番地をモニタなどで書き換えるだけでnクイーンの解が求められます。

15行：プログラム中で使用する（破壊する）レジスタは、スタックに退避して置くことが望まれます。これをしないと、このプログラムを呼び出すプログラムの方で予期せずにレジスタの値がかわってしまい誤動作したり、バグの原因になったりします。

18行他：この様にプログラムの切れ目に“＊”による注釈文（空行でも良い）を入れておくとプログラムがみやすくなります。

65行：スタックに退避したレジスタを元に戻します。ここは、

```
PULS D,X
RTS
```

としても良いのですが、同じ動作をする

```
PULS D,X,PC
```

の方が速くて短くてすみます。

72行：ANSWERというルーチンがここから始まることを示します。もちろん次の行と合わせて、

```
ANSWER PSHS D,X
```

としてもかまいません。

77行：＄B615からのDレジスタを10進で出力するF—BASICのROM内のルーチンを呼び出します。このようにROM内ルーチンを用いると便利です。

78行：シングルクオート（'）を用いると次の文字のアスキーコードが値となります。この例では、“＝”のアスキーコードは＄3Dなので、

```
LDA #'=
```

は

```
LDA #$3D
```

と同じです。

79，81行：Aレジスタに格納されている数をアスキーコードとする文字として出力するROM内ルーチンを呼び出します。

88行：改行を行うROM内ルーチンを呼び出します。

91行～95行：作業領域を定義します。
96行：NQUEENの数だけ領域を確保します。
98行：プログラムの最後には END 命令をおきます。実行開始はSTARTからにします。

だいたい以上のようになっています。このプログラムを実行してみたのが図A―16です。

もし実行しても表示されないときは、暴走してしまったと思われます。次の順序で処置してください。

①ブレークキーを押しながらリセットスイッチを押してください。これで、“Ready”が出力されたら、プログラムは残っています。CLEAR文できちんと領域を確保したかどうかを確認してください。

②①でだめだったときには、あきらめてリセットスイッチだけを押してください。残念ながらプログラムは残っていませんから、もう一度確認しながら再実行してください。

③実行手順が悪くなければ、プログラムに間違いがあったと考えられます。アセンブルリストをきちんと確認してください。この時、特に、PSHSした後の PULS し忘れなどをよく確認してください。これはすぐに暴走に結びつきます。

図A―17に、このプログラムをnクイーンとして実行したときの結果を示します。これを参考にしてさらに速いアルゴリズムを開発してみるのも、良い勉強法だと思います（図A―18のBASICプログラムを使用して実験・時間には解の表示時間が含まれています）。

クィーンの数(n)	解の数	実行時間(分：秒)
4	2	00：00
5	10	〃
6	4	〃
7	40	00：01
8	92	00：04
9	352	00：19
10	724	00：59
11	2680	04：50
12	14200	28：53

図A―19

〔図A－15　アセンブルリスト〕

```
 1                          *******************
 2                          *
 3                          * N-QUUEN ノ カイ ヲ
 4                          *   モトメル プログラム
 5                          *
 6                          *******************
 7                          *
 8  5000                            ORG    $5000     プログラム ハ $5000バンチ カラ
 9                          *
10                  0008    NQUEEN  EQU    8         QUEEN ノ カズ
11                          *
12  5000 20   01            START   BRA    MAIN
13  5002 08                 DIM     FCB    NQUEEN    QUEEN ノ カズヲ ストア シテオク
14                          *
15  5003 34   16            MAIN    PSHS   D,X
16  5005 CC   0000                  LDD    #0        カウンタ ヲ クリア
17  5008 FD   508F                  STD    COUNT
18                          *
19  500B BE   5095                  LDX    #BOARD
20  500E B7   5091                  STA    XPOINT
21                          *
22  5011 7F   5092          NEXTX   CLR    YPOINT    1ギョウ メ カラ
23                          *
24  5014 FC   5091          CHECK   LDD    XPOINT    CHECKヨウ ノ ショキカ
25  5017 F7   5093                  STB    YCHEC1
26  501A F7   5094                  STB    YCHEC2
27  501D 20   17                    BRA    CHECK2
28                          *
```

```
 29   501F 7A   5093      CHECK1  DEC    YCHEC1
 30   5022 7C   5094              INC    YCHEC2
 31   5025 E6   86                LDB    A,X
 32   5027 F1   5092              CMPB   YPOINT    ﾖｺ ﾉ ﾁｪｯｸ
 33   502A 27   28                BEQ    NEXTY
 34   502C F1   5093              CMPB   YCHEC1    ﾋﾀﾞﾘｳｴ ﾉ ﾁｪｯｸ
 35   502F 27   23                BEQ    NEXTY
 36   5031 F1   5094              CMPB   YCHEC2    ﾋﾀﾞﾘｼﾀ ﾉ ﾁｪｯｸ
 37   5034 27   1E                BEQ    NEXTY
 38                       *
 39   5036 4A             CHECK2  DECA             ﾋﾀﾞﾘ ﾉ ﾚﾂ ﾍ
 40   5037 2A   E6                BPL    CHECK1
 41                       *
 42   5039 FC   5091              LDD    XPOINT    ｼﾞｯｻｲﾆ ｵｸ
 43   503C E7   86                STB    A,X
 44                       *
 45   503E 4C                     INCA             ﾂｷﾞﾉ ﾚﾂ ﾍ
 46   503F B7   5091              STA    XPOINT
 47   5042 B1   5002              CMPA   DIM
 48   5045 26   CA                BNE    NEXTX
 49                       *
 50   5047 8D   1D                BSR    ANSWER    ｺﾀｴ ﾐｯｹ
 51                       *
 52   5049 7A   5091      BACKX   DEC    XPOINT    ﾏｴﾉ ﾚﾂ ﾍ
 53   504C B6   5091              LDA    XPOINT
 54   504F E6   86                LDB    A,X
 55   5051 F7   5092              STB    YPOINT
 56                       *
 57   5054 7C   5092      NEXTY   INC    YPOINT    ﾂｷﾞﾉ ｷﾞｮｳ ﾍ
 58   5057 F6   5092              LDB    YPOINT
 59   505A F1   5002              CMPB   DIM
 60   505D 26   B5                BNE    CHECK
 61                       *
 62   505F 7D   5091              TST    XPOINT    1ｷﾞｮｳﾒ ｶﾉ ﾁｪｯｸ
 63   5062 26   E5                BNE    BACKX
 64                       *
 65   5064 35   96                PULS   D,X,PC    ｾﾞﾝﾌﾞ ｵｰﾜﾘ
 66                       *
 67                       ***************
 68                       *
 69                       * ｶｲ ｦ ｼｭﾂﾘｮｸ
 70                       *
 71                       ***************
 72                 5066  ANSWER  EQU    *
 73   5066 34   16                PSHS   D,X
 74   5068 FC   508F              LDD    COUNT     ｶｳﾝﾀ ｦ 1 ﾌﾔｽ
 75   506B C3   0001              ADDD   #1
 76   506E FD   508F              STD    COUNT
 77   5071 BD   B615              JSR    $B615     Dreg ｦ 10ｼﾝ ﾃﾞ ｼｭﾂﾘｮｸ
 78   5074 86   3D                LDA    #'=
 79   5076 BD   D08E              JSR    $D08E     Areg ﾉ ｷｬﾗｸﾀ ｦ ｼｭﾂﾘｮｸ
 80   5079 5F                     CLRB             ﾙｰﾌﾟ ｶｳﾝﾀ
 81   507A 8E   5095              LDX    #BOARD
 82   507D A6   85        ANSLOP  LDA    B,X
 83   507F 8B   31                ADDA   #'1       0-7 ｦ '1'～'8' ﾆ ﾍﾝｶﾝ
 84   5081 BD   D08E              JSR    $D08E     Areg ﾉ ｷｬﾗｸﾀ ｦ ｼｭﾂﾘｮｸ
 85   5084 5C                     INCB
 86   5085 F1   5002              CMPB   DIM
 87   5088 26   F3                BNE    ANSLOP
 88   508A BD   9B50              JSR    $9B50     ｶｲｷﾞｮｳ(CR&LF) ｦ ｽﾙ
 89   508D 35   96                PULS   D,X,PC
 90                       *
 91   508F                COUNT   RMB    2         ｶｲ ﾉ ｶｽﾞ
 92   5091                XPOINT  RMB    1         ｶﾃｲ ｽﾙ ﾚﾂ ﾉ ｻﾞﾋｮｳ
 93   5092                YPOINT  RMB    1         ｶﾃｲ ｽﾙ ｷﾞｮｳ ﾉ ｻﾞﾋｮｳ
 94   5093                YCHEC1  RMB    1         ﾐｷﾞｳｴ ﾁｪｯｸ ﾖｳ
 95   5094                YCHEC2  RMB    1         ﾐｷﾞｼﾀ ﾁｪｯｸ ﾖｳ
 96   5095                BOARD   RMB    NQUEEN    ﾊﾞﾝ ﾉ ﾃﾞｰﾀ ｦ ｵｸ ﾄｺﾛ
```

```
  97                        *
  98                                    END     START     ﾌﾟﾛｸﾞﾗﾑ ﾊ ｺｺﾃﾞ ｵｼﾏｲ

O ERROR(S) DETECTED

SYMBOL TABLE:

ANSLOP 507D   ANSWER 5066   BACKX  5049   BOARD  5095   CHECK  5014
CHECK1 501F   CHECK2 5036   COUNT  508F   DIM    5002   MAIN   5003
NEXTX  5011   NEXTY  5054   NQUEEN 0008   START  5000   XPOINT 5091
YCHEC1 5093   YCHEC2 5094   YPOINT 5092
```

〔図A－16 実行例〕

```
LOADM "QUEEN"

Ready
EXEC &H5000
1=15863724
2=16837425
3=17468253
4=17582463
5=24683175
6=25713864
7=25741863
8=26174835
9=26831475
10=27368514
11=27581463
12=28613574
13=31758246
14=35281746
15=35286471
16=35714286
17=35841726
18=36258174
19=36271485
20=36275184
21=36418572
22=36428571
23=36814752
24=36815724
25=36824175
26=37285146
27=37286415
28=38471625
29=41582736
30=41586372
31=42586137
32=42736815
33=42736851
34=42751863
35=42857136
36=42861357
37=46152837
38=46827135
39=46831752
40=47185263
41=47382516
42=47526138
43=47531682
44=48136275
45=48157263
46=48531726
47=51468273
48=51842736
49=51863724
50=52468317
51=52473861
52=52617483
53=52814736
54=53168247
55=53172864
56=53847162
57=57138642
58=57142863
59=57248136
60=57263148
61=57263184
62=57413862
63=58413627
64=58417263
65=61528374
66=62713584
67=62714853
68=63175824
69=63184275
70=63185247
71=63571428
72=63581427
73=63724815
74=63728514
75=63741825
76=64158273
77=64285713
78=64713528
79=64718253
80=68241753
81=71386425
82=72418536
83=72631485
84=73168524
85=73825164
86=74258136
87=74286135
88=75316824
89=82417536
90=82531746
91=83162574
92=84136275

Ready
```

〔図A－18 実験用プログラム〕

```
10 CLEAR ,&H5000
20 INPUT "N=",N
30 POKE &H5002,N
40 TIME$="00:00:00"
50 EXEC &H5000
60 PRINT TIME$
```

第 $ B 章

BIOS

——入出力の基礎——

1. BIOSの概略

既に述べましたが、BIOSは機種間のI／Oルーチンの違いを吸収して、プログラムの共有化を計るための一種のサブルーチンパッケージです。このBIOSは、FM—7シリーズの数あるI／O装置の入出力を一手に受け請ってくれる便利なパッケージです。サポートしているI／O装置は、

①アナログ入力ポート
②オーディオカセット
③ブザー
④漢字ROM
⑤プリンタ
⑥ディスク
⑦ディスプレイ・サブシステム
⑧バブルカセット

の8つです。このうち①⑧はなじみのない方も多いと思います。この①⑧はFM—8時代の名残りで、互換性を保つためにサポートされています。

⑦は既に述べたサブCPUを意味しています。このBIOSを用いればサブCPU（すなわちディスプレイサブシステム）をGDC（グラフィックディスプレイコントローラの略。グラフィックを簡単に処理してくれるLSIのこと）にみたてて使うことができます。

2. BIOSの使い方

さて、それでは実際にBIOSを使ってI／O装置を制御するにはどうしたら良いのでしょうか。

システム仕様にあるとおり

①RCB領域としてRAM上に8バイトの領域を確保する。

②RCBにリクエスト番号を設定する。

③RCBに必要なパラメータをすべて設定する。

④XレジスタにRCBの先頭アドレスをセットする。

⑤BIOSをサブルーチンコールする。このサブルーチンコールは

```
JSR [$FBFA]
```

で行います。

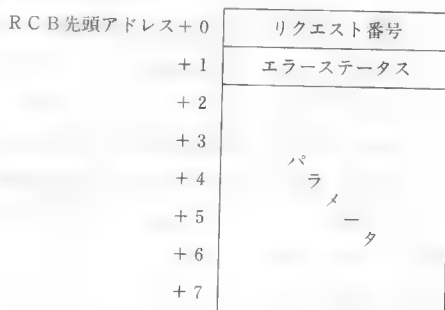
という順序で実行します。といってもこれではわからないと思いますので、1ステップずつ解説することにします。

◀第1ステップ▶

このBIOSはプログラミングをやり易くするために、BIOSの呼び出し番地（サブルーチンのアドレス）を1箇所に集中してあります。ですから、要求する仕事の内容が漢字ROMのことであってもDISKのことであっても、呼び出し番地にかわりはなく、すべて⑤に示した命令を用いることになっています。

こうなると、要求する仕事の内容をBIOSに知らせる必要があります（もし、仕事の内容によって呼び出し番地が違っていれば、呼び出す番地によりおのずと仕事の内容は決まります)。そうでなければBIOSは仕事の内容がわからず、立ち往生してしまいます。その仕事の内容を区別するのがリクエスト番号です。

また仕事には細かいパラメータ（制御データ）が伴います。例えばカセットのモーターをコントロールする場合には、モーターをONにするのかOFFにするのかをパラメータとして与えることが必要となるわけです。

このリクエスト番号とパラメータを与えるのに用いるのが、RCBと呼ばれるものです。このRCBというのは、Request Control Block（リクエスト コントロール ブロック）の略で、BIOSを使用するときには必ずなければいけないものです。RCBはメモリ上の任意の位置に置くことができますが、8バイトの大きさを必要とし、図B―1のような構成となっています(詳しくは追って解説します)。

RCB先頭アドレス	
+0	リクエスト番号
+1	エラーステータス
+2	パラメータ
+3	
+4	
+5	
+6	
+7	

図B－1　RCBの構成

ですからBIOSを使用する際にはRCB領域を確保しておく必要があります。確保するには、作業領域に

```
RCB  RMB  8
```

として8バイト確保するのが一般的です。この他にもFCB命令等を用いる方法もありますが、それについては後述します。

このRCB領域の確保だけは、プログラミングの際に行うので、実行時に行う②以降のステップとは、ちょっと毛色が違います（もっとも高度なテクニックを用いれば、実行時にRCB領域を確保する方法もないわけではありません)。

◀第2ステップ▶

既に述べたようにBIOSに仕事の内容を区別するのにはリクエスト番号を用います。このリクエスト番号は、常にRCB領域の最初のバイト(先頭番地）に格納することになっています。このリクエスト番号は、図B―2にあげています。ここではそれぞれのリクエストの動作については解説しませんが、例えばMOTORと名前のついた仕事をやってもらいたいのであれば、リクエスト番号

16進	10進	ラベル名	16進	10進	ラベル名
$00	0	ANALGP	$0E	14	LPOUT
$01	1	MOTOR	$0F	15	HDCOPY
$02	2	CTBWRT	$10	16	SUBOUT
$03	3	CTBRED	$11	17	SUBIN
$04	4	INTBBL	$12	18	INPUT
$05	5	SCREEN	$13	19	INPUTC
$06	6	WRTBBL	$14	20	OUTPUT
$07	7	REDBBL	$15	21	KEYIN
$08	8	RESTOR	$16	22	KANJIR
$09	9	DWRITE	$17	23	LPCHK
$0A	10	DREAD	$18	24	BIINIT
$0B	11		$19	25	
$0C	12	BEEPON	$1A	26	
$0D	13	BEEPOF	$1B	27	

図B－2　BIOSリクエスト番号表

は1ですから、

```
LDA #1  ←リクエスト番号
STA RCB
```

とすれば良いわけです。もちろん、ポジションインディペンデントに

```
STA RCB PCR
```

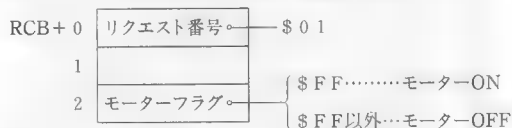
としても構いません。

◀第3ステップ▶

第3ステップは**パラメータの設定**です。パラメータはＲＣＢ領域の3バイトめから8バイトめ(先頭番地＋２～先頭番地＋７）までに設定することになっています。このパラメータは、ＢＩＯＳに要求する仕事の種類、すなわちリクエストの違いによって設定する値の意味が違っているので、それぞれのリクエストの使用法を参照しなければいけません。使用法は、システム仕様か解析マニュアルに載っています。

それでは先程のＭＯＴＯＲ（このリクエストはカセットレコーダのモータを制御するリクエストです）を例に取って解説しましょう。

図Ｂ—3は解析マニュアル基礎編のＭＯＴＯＲの項です。機能の項に、このリクエストの仕事の内容が述べられています。パラメータの項が今私たちが調べようとしていることです。第2ステップで述べたリクエスト番号1をＲＣＢ＋０、すなわち、ＲＣＢの先頭番地に設定しなければいけないことがわかります。次のＲＣＢ＋１には何も書かれていません。これは、その番地には何も設定する必要がないことを示しています。

オーディオカセットのモーターをコントロールします。
パラメータ：Ｘレジスタ←ＲＣＢ先頭アドレス

このリクエストはＲＣＢ領域としては3バイトしか必要としません。

モーターフラグの値によってリモート端子をコントロールします。このリクエストではエラーは生じません。

図Ｂ－３　BIOS：MOTOR

その次のＲＣＢ＋２が狭い意味でのパラメータです（広義としては、リクエスト番号を含めてＲＣＢ内に設定する値全てを表します)。ここではモーターフラグとなっています。右にその意味が書いてあるとおり、モーターをＯＮにしたいのか、ＯＦＦにしたいのかを、このパラメータでＢＩＯＳに教えてあげています。モーターをＯＮにしたいのであれば、＄ＦＦを設定します。

```
LDA #$FF
STA RCB+2
```

とすれば良いわけです。

ここで初めての表現が出てきたので解説しましょう。これまでオペランドには、(アドレッシングモードなどを指定する、“ＰＣＲ”などの他には）数値（10進、16進、２進）かシンボルを１つだけ書いてきました。ところが実は、この部分には任意の式を書くことができるのです（アセンブラによってはこの機能がないものもあります。富士通やＴＳＣ社のものにはついています)。ですから、ＲＣＢ＝＄１２３４だったとすれば、

```
STA RCB+2
```

は

```
STA $1234+2
```

であり、結局

```
STA $1236
```

と同じになります。この式で用いることのできる演算子は“＋”の他に“－”、“＊”、“／”などがあります（富士通やＴＳＣ社のものは数多くの演算子をサポートしています。優先順位なども含めて詳しいことはマニュアルを参照してください)。

例に戻って、図Ｂ—3の中にあるようにリクエストによってはＲＣＢ領域として８バイト全部必要ではないものもあります。ＭＯＴＯＲでは3バイトしか必要としません。この場合、残りの５バイトに値を設定する必要は全くありません。また、ＭＯＴＯＲなどのように使用するＲＣＢ領域のバイト数が確定している場合には、そのバイト数だけＲＣＢを確保しなくても問題ありません。例えばＭＯＴＯＲであれば

```
RCB  RMB  3
```

でも良いわけです.

以上でＲＣＢ領域内の設定は終了しました。

◀第4ステップ▶

さて、RCB領域はメモリ上のどこにおいても良いことはすでに述べたとおりです（どこにでも、といってもフリーエリア内でなければいけません)。しかしBIOSの立場に立ってみればどこでも良いというのは非常に困ったことです。どこか一定のアドレスであれば、どこにあるのかがわかるので良いのですが、どこでもよいとなると、BIOSにはRCB領域の場所がわかりません。これではまずいので、私たちがBIOSにその場所を教えてやらなければいけません。

そのために用いるのがXレジスタです。Xレジスタに RCB 領域の先頭アドレスをいれるわけです。すなわち

```
LDX #RCB
```

または

```
LEAX RCB, PCR
```

としてやれば良いわけです。こうしてやればBIOSは、Xレジスタを用いてRCB領域のパラメータを悩むことなく参照できることになります。

ところで、このXレジスタにRCB領域の先頭アドレスを設定する動作は、第3ステップのパラメータの設定においてXレジスタを使用しない限り、第2ステップ（リクエスト番号の設定）の前に行っても構いません。すなわち第4、第2、第3ステップの順で行っても良いのです（BIOSの側にしてみれば、XレジスタとRCB領域がきちんと設定されてさえいれば、設定の順序は問題とならない)。「しても良い」という表現をしましたが、実はこのように行われるのが普通です。というのは、最初に

```
LDX #RCB
```

としておけば、リクエスト番号の設定は

```
LDA #1      ←リクエスト番号
STA ,X
```

パラメータの設定は

```
LDA #$FF
STA 2,X
```

とすることができ、この方がプログラムの長さや実行速度の点で有利だからです。またこの方式だとRCBの何バイトめにパラメータを設定するのかが明確になるという点も見逃せません（本書でも特に理由のない限り、この方式を用いることにします)。

◀第5ステップ▶

BIOS呼び出しの下準備が全てそろったところで、いよいよBIOSを呼び出す方法について解説します。といってもそれ程大変ではありません。

```
JSR [$FBFA]
```

とすれば良いのです。もしいろいろなところでBIOSを呼ぶのであれば

```
BIOS EQU $FBFA
```

としておいて

```
JSR [BIOS]
```

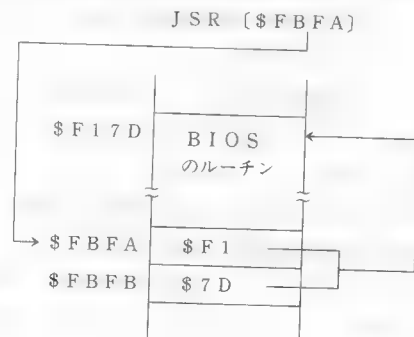
とすると良いでしょう。

この命令を実行した時の動作を少し考えてみましょう。エクステンデッドインダイレクトですから、まず、$FBFA、$FBFB番地を参照します。ここはROMになっておりここに実際のBIOSのルーチンのアドレスが格納されています。ここまでは、FM―8↔FM―7↔FM―11の間で共通です。機種によって違っているのは、$FBFA、$FBFB番地に格納されている数値です。これがFM―7シリーズの場合には$F17Dとなっています（FM―8では、$F2D8となっています)。ですから、

```
JSR [$FBFA]
```

はFM―7では

```
JSR $F17D
```

と同じというわけです。BIOSを呼び出すのに

図B－4　BIOSの呼び出し

この＄Ｆ１７Ｄを用いても良いのですが、こうするとFM—８やFM—11では動作しなくなります。ソフトの共有化を計るというＢＩＯＳの精神のからもこれは避けて、きちんと

```
JSR [$FBFA]
```

を使用しましょう。

◀第６ステップ▶

第５ステップまででＢＩＯＳの実行は終了しました。しかし本当にきちんとリクエストをこなしてくれたのでしょうか。このＢＩＯＳはＩ／Ｏ装置を扱っている関係上、エラーが生じることがあります（例えばディスクのリードエラーなどです)。エラーが生じるとＢＩＯＳはそこで処理を中断して呼び出したルーチンに戻ってきてしまいます。

このようにエラーが生じる可能性がある（そのエラーの原因が私たちにあるとしても、機械の方にあるとしても）以上、私たちはエラーがおきたのか、おきなかったのかをなんらかの方法でみわけなければいけません。このために存在するのが、ＲＣＢ領域の２バイトめ（ＲＣＢ＋１番地）です。ＲＣＢ領域には、私たちからＢＩＯＳへ情報を伝えるという役割とこの逆の２つの役割があります。このＲＣＢ領域の２バイトめはまさに後者のために設けられているのです。

ＢＩＯＳは、リクエストの処理中にエラーが生じた場合には、ＲＣＢ領域の２バイトめ（ＲＣＢ＋１）にエラー番号を格納します。また、エラーが生じずに、正常にリクエストの処理を終了した時には、ここに０が格納さます。ですから例えば、エラーがあったときにＥＲＲＯＲに分岐したいとすれば、

```
JSR [$FBFA]
LDA 1,X  ← TST でも良い
BNE ERROR
```

とします（ＢＩＯＳの呼び出しでは、ＣＣレジスタ以外のレジスタは保存されています。つまり、JSR の前後ではＣＣレジスタを除いて同じ値を持っていることになります)。

同時にＢＩＯＳは、エラーが生じたときにはＣＣレジスタ中のＣフラグをセット（“1”）し、エラーが生じなかったときにはＣフラグをリセット（“0”）してくれます。ですから先の例は

```
JSR [$FBFA]
BCS ERROR
```

でも良いわけです。

先に「ＢＩＯＳではエラーが生じる可能性がある」と書きましたが、中には（ブザーやモーター関係のように）決してエラーを生じないリクエストもあります。もし使用するリクエストがエラーを生じないものであれば、エラー処理のルーチンへ分岐する命令（前例のＢＣＳなど）は必要ありません。しかし、使用するリクエストがエラーを生じる可能性があるのでしたら、前例のようにしてエラー処理へ分岐する命令を入れておいた方がよいでしょう。そして、エラー処理を行うところでは、もう一度ＢＩＯＳを呼び出してみたり（リトライ：再試行）するとよいでしょう。

◀第７ステップ▶

さて、ＢＩＯＳのリクエストには、私たちからＢＩＯＳに値を与えて処理してもらうだけではなく、ＢＩＯＳを通して入力装置から値を受け取るリクエストもあります。例えば、キー入力を受け取るＫＥＹＩＮルーチンなどはその一つです。こういったリクエストによって得られた値（復帰情報）はＢＩＯＳによって所定の所に格納されます。例えば、カセットから１バイトのデータを読み出すＣＴＢＲＥＤというリクエストの場合、得られた値（データ）はＲＣＢ領域の３バイトめ（ＲＣＢ＋２）に格納されます。ですから私たちは、

```
JSR [$FBFA]
```

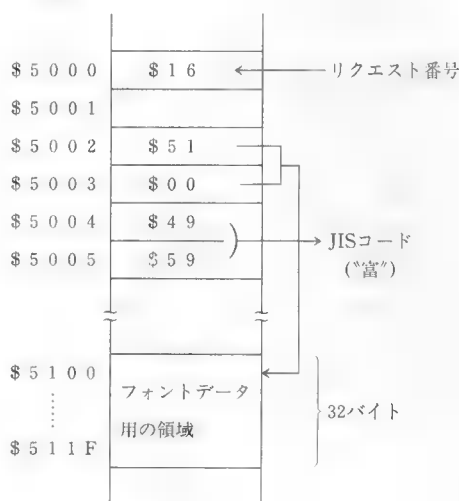
の後、

```
LDA 2,X
```

で、望んでいたデータを得ることができるというわけです。どこにデータが格納されるかは、各リクエストの解説をシステム仕様または解析マニュアル基礎編で確認してください。

ところで、ＢＩＯＳのＲＣＢ領域は最大でも８バイトでした。しかしリクエストによっては、パラメータやＢＩＯＳが返してくる値のバイト数がこのＲＣＢ領域では全く不足するというものがあります。このようなリクエストではＲＣＢ領域内

に別の領域の先頭アドレスを格納するという方法を取っています。

例として漢字ROMのフォントデータ（文字のドットのデータ）を読み出すKANJIRというリクエストを取り上げます。フォントデータは32バイトあるため、RCB領域には収まりません。そこで、RCB＋2、RCB＋3番地にRCB領域とは別に確保した32バイトの領域の先頭アドレスを格納します。RCB領域が＄5000番地から、フォントデータ用の領域が＄5100番地からとすれば図B—5のようにRCB領域を設定するわけです。こうすることによってBIOSは＄5100番地からフォントデータ用の領域であることを理解して、指定した漢字のフォントデータを＄5100番地からの32バイトに格納してくれます。

図B－5 特殊なRCBの設定

3. *BIOSの構造*

前節でかなり詳しくBIOSの使い方を述べましたが、ここまでくるとBIOSの中身はどうなっているかと気になる人もいると思います。そこでこの節では、そのBIOSのドライバルーチン（各リクエスト別にふりわけたりする部分）を題材にして、その中に用いられているテクニックなどに言及してみることにします。

図B—6がBIOSのドライバルーチンの逆アセンブル（Dis Assemble）リストです。逆アセンブルというのは、アセンブルの逆でマシンコードからアセンブリ言語を得ることをいいます。逆アセンブルするには逆アセンブラというプログラムを使用します。図B—6は巻末の付録に掲載した逆アセンブラを用いて出力したものです。

既に述べましたが、BIOSの呼び出しの

```
JSR [$FBFA]
```

は、FM—7シリーズの場合

```
JSR $F17D
```

と同じです。ですから＄F17Dからのプログラムを解析すればよいということになります。それでは1ステップずつ解説していきましょう。

①：ここでは、すべてのレジスタをSスタック上に退避しています。これによりBIOSルーチン内ではSレジスタを除くレジスタを破壊してもよいことになります。

②③：DPレジスタに＄FDを代入します。なぜDPを設定するかというと、BIOSではプログラムの性格上、I／O領域（＄FD00～＄FDFF：9章1節参照）を参照することが多くなります。そこで＄FDをDPレジスタに設定すれば

```
LDA $FD00    (3バイト)
```

などが

```
LDA <$00    (2バイト)
```

となりこれはダイレクトモードを用いることによって速くかつ短くなるからです。

④⑤：ここではRCB領域にセットされているリクエスト番号をBレジスタにロードし、さらにAレジスタをクリアします。このことにより結局Dレジスタにリクエスト番号がセットされたことになります。

⑥⑦：Bレジスタを左シフト、Aレジスタを回転（ローテート）しています。これはDレジスタにはリクエスト番号×2がセットされることになります。

⑧：Yレジスタに＄F1A9をセットします。この＄F1A9はBIOSの各リクエストの処理ルーチンのエントリアドレス（入口番地）

が格納されているジャンプテーブルの先頭アドレスです。

ジャンプテーブルは＄Ｆ１Ａ９から＄Ｆ１Ｅ０までにあって図Ｂ—7のようになっています。各リクエストごとに２バイトでそのリクエストの処理ルーチンのエントリアドレスを示しています。例えば、＄Ｆ１ＡＢ、＄Ｆ１ＡＣ番地の＄Ｆ２、＄Ａ３は、リクエスト番号１のリクエストの処理をするルーチンがあるアドレス＄Ｆ２Ａ３を示しています。

⑨：Ｙ←Ｄ＋Ｙを行います。すなわち、ジャンプテーブルの先頭アドレスにリクエスト番号×２を加えることにより、ジャンプテーブル中の呼び出すべきリクエストのエントリアドレスが格納されているアドレスを得ます。例えば、リクエスト番号が２だったとすれば，Ｄレジスタは４、Ｙレジスタは＄Ｆ１Ａ９だったので、これよりＹレジスタが＄Ｆ１ＡＤとなります。この＄Ｆ１ＡＤはリクエスト番号２のリクエストの処理ルーチンエントリアドレス＄Ｆ２Ｂ０が格納されているアドレスを示しています。

⑩⑪⑫：これは⑨で得られたＹレジスタの値が許された範囲を超えていないかをチェックする部分です。範囲を超えたとき（リクエスト番号が28以上だとＹレジスタが＄Ｆ１Ｅ０以

〔図Ｂ－６　ＢＩＯＳのドライバルーチン〕

```
F17D-34 7F          ① PSHS   CC,A,B,DP,X,Y,U
F17F-86 FD          ② LDA    #$FD
F181-1F 8B          ③ TFR    A,DP
F183-E6 84          ④ LDB    ,X
F185-4F             ⑤ CLRA
F186-58             ⑥ ASLB
F187-49             ⑦ ROLA
F188-10 8E F1 A9    ⑧ LDY    #$F1A9
F18C-31 AB          ⑨ LEAY   D,Y
F18E-86 02          ⑩ LDA    #$02
F190-10 8C F1 DF    ⑪ CMPY   #$F1DF
F194-22 06          ⑫ BHI    $F19C
F196-34 10          ⑬ PSHS   X
F198-AD B4          ⑭ JSR    [,Y]
F19A-35 10          ⑮ PULS   X
F19C-35 01          ⑯ PULS   CC
F19E-A7 01          ⑰ STA    $01,X
F1A0-26 03          ⑱ BNE    $F1A5
F1A2-1C FE          ⑲ ANDCC  #$FE
F1A4-8E 1A 01       ⑳ LDX    #$1A01
F1A7-35 FE          ㉑ PULS   A,B,DP,X,Y,U,PC
```

〔図Ｂ－７　ＢＩＯＳジャンプテーブル〕

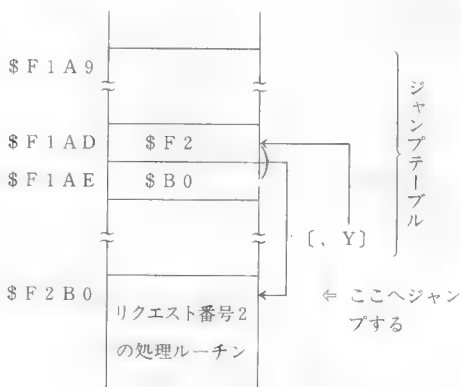
```
      ┌……………………………………リクエスト番号０のリクエストの処理ルーチンエントリアドレス
F1A9- [F1 E1] : 円ﾋ  F1AB- F2 A3 : 年｣
F1AD- F2 B0 : 年ｰ    F1AF- F3 30 : 月0
F1B1- F4 C8 : 日ﾈ    F1B3- F9 98 : 市ｒ
F1B5- F5 29 : 時)    F1B7- F5 29 : 時)
F1B9- FE 02 : ※.    F1BB- FE 05 : ※.
F1BD- FE 08 : ※.    F1BF- F5 A8 : 時ｨ
F1C1- F5 A1 : 時｡    F1C3- F5 A4 : 時､
F1C5- F5 AA : 時ｪ    F1C7- F7 B8 : 秒ｸ
F1C9- F5 F1 : 時円   F1CB- F5 F1 : 時円
F1CD- F6 55 : 分U    F1CF- F7 13 : 秒.
F1D1- F7 2D : 秒-    F1D3- F7 8F : 秒┼
F1D5- F2 1F : 年.    F1D7- F5 C7 : 時ﾇ
F1D9- FB 05 : 町.    F1DB- F5 A8 : 時ｨ
F1DD- F5 A8 : 時ｨ    F1DF- [F5 A8] : 時ｨ
                           └……リクエスト番号２７（＄１Ｂ）のリクエストの処理ルーチンエントリアドレス
```

上になる）にはAレジスタに2を入れて＄F19C（⑯）に分岐します。この時のAレジスタの2は、BIOSのエラー番号2 (Device Unavailable エラー）の2を示しています。範囲を超えていないときにはそのまま⑬へ。

⑬⑭⑮：各リクエストの処理ルーチンを呼び出します。ここではインダイレクトを用いて巧妙に処理しています。文章ではわかりにくくなると思いますので図B—8に図示しておきます。Yレジスタには要求するリクエストの処理ルーチンのエントリアドレスが格納されているアドレスが入っている点に注意してください。

さらに、このJSRの前後では保存しておきたいXレジスタを退避し復帰するということをしています。(XレジスタにはRCB領域の先頭アドレスが格納されたままになっています)。

⑯：ここでは①でスタックに退避したレジスタ群のうちCCレジスタだけを復帰します。これは，CCレジスタ中のCフラグにBIOSでのエラーの有無をセットするためです。

⑰：各リクエストの処理ルーチンは、JSRで呼ばれた後、各処理を行い、最後にAレジスタにエラー番号をセットして戻ってきます(エラーがなければ0をセットします）このAレジスタのエラー番号をRCB領域の2バイトめ（RCB＋1すなわちX＋1）にセットするわけです。

図B−8 JSR〔，Y〕の動き

⑱：もしエラーがあると、⑰によってZフラグがリセットされ、エラーがなければZフラグがセットされます（STA命令ではN、Zフラグが設定されます。）これにより、Zフラグがリセット（＝エラーあり）されているならばBNE命令（Zフラグ＝0で分岐）によって＄F1A5に分岐します。

⑲：＄FE＝%11111110でCCレジスタをANDすることにより、Cフラグ(最下位：bit 0）をクリア（0にする）します。

⑳：この部分がこのルーチンの中で一番高度なテクニックを用いている箇所です。ちょっと⑱に戻りますが、⑱ではエラーがあった時に＄F1A5番地に分岐していました。この＄F1A5番地というのは、左側のダンプリストをみるとわかると思いますが、

```
LDX #$1A01
```

のマシンコード

```
$8E $1A $01
```

の＄1Aのあるアドレスです。＄F1A5番地から逆アセンブルすると、

```
F1A5-1A 01 ORCC #$01
```

となり、＄01でORすることによりCCレジスタ中のCフラグをセットします（1にする)。

このようにXレジスタにロードする＄1A01という定数には定数としての意味はなく、

```
ORCC #$01
```

という命令としての意味を持っています。すなわち、LDX #を意味する＄8Eはそれに続く2バイトの命令を飛び越す(実行しない）ようにするためのものなわけです。

このようなテクニックは高度なプログラムではよく用いられる方法です。でも最初のうちはこういったテクニックは用いずに、

```
        BNE    ラベル1
        ANDCC  #$FE
        BRA    ラベル2
ラベル1  ORCC   #$01
ラベル2  PULS   ……
```

する方がよいでしょう。

㉑：ここでは①で退避したレジスタ群を復帰し

ています。⑯で既に復帰したCCレジスタは含みません。そして、ここでは、PC（プログラムカウンタ）も同時に復帰する（これはBIOSが呼び出されたときに戻り番地としてスタックに積んであったもの）ことにより、レジスタ群の復帰とこのルーチンからの戻りとを同時に行います。

以上がBIOSのドライバルーチンの構造です。詳しく述べたのでかえってわかり難いかもしれません。「プロの人はこんなプログラムを組むのだなあ」と思っていただければ十分です。

4. ブザー制御を例に

さて、ここまででBIOSの使い方と構造とを追ってきました。そこでここではその使い方を実践して理解を深めることにしましょう。

図B―9をみてください。これはリクエスト番号＄0CのBEEPONのリクエストを用いた例です。2節の第1ステップが7行め、第2ステップが3～4行め、第3ステップはセットすべきパラメータがないのでパス、第4ステップが2行め、第5ステップが5行めに対応しています。このリクエストはエラーを生じませんし、値を返してもきませんから第6、第7ステップはありません。

ところでこのサンプルプログラムは基礎編で数多く出てきたサンプルプログラムとちょっと違った終了のしかたをしています。基礎編では

JMP　＄ABF4

でモニタへとんで終了しましたが、ここでは

RTS

で終っています。この違いは、このプログラムの実行開始の方法によります。基礎編のプログラムはすべてモニタからGコマンドで実行させてきました。Gコマンドでは，JMP命令でプログラムを実行し始めるために終了もJMP命令で

JMP　＄ABF4

としなければならないわけです。

ところがBASICのEXEC命令によりプログラムを実行させる場合にはJMP命令による終了はしません。これは，EXEC命令の場合にはBASICインタプリタ内部からJSR命令でプログラムを実行し始めるためです。JSR命令で実行される以上、作るプログラムは形式上サブルーチンの形をしていなければなりません。そのため、終りにはRTS命令をおいて、サブルーチンから戻るようにしなければならないわけです。

ちょっと話がそれましたが、要するに図B―9のプログラムはBASICから

EXEC　&H5000

で実行しなければいけないということです。ではアセンブルリスト左側に出力さているマシンコードをモニタで入力して確認してください。その上でEXEC命令で実行してみてください。

ブザーがなり始めて“Ready”が表示されたことと思います（異常があれば再度プログラムを確認してください)。ブザーを止めるには、BASIC

〔図B－9　BEEPONリクエスト〕

```
 1   5000                        ORG    $5000
 2   5000 8E    500C             LDX    #RCB
 3   5003 86    0C               LDA    #$0C
 4   5005 A7    84               STA    ,X
 5   5007 AD    9F FBFA          JSR    [$FBFA]
 6   500B 39                     RTS
 7   500C              RCB       RMB    8
 8                               END

0 ERROR(S) DETECTED

SYMBOL TABLE:

RCB      500C
```

でＢＥＥＰ０とするか、ブレークキーを押してください。

これが一番簡単なＢＩＯＳの使い方といえるのでその内容については省略しても良いでしょう(わからなければ，このプログラムを見ながらこの章の２節を読み返してみてください。)

ただ鳴らすだけではあまり面白くないので効果音的な音を出す実験をしてみます。まずは、下準備の実験です。

図Ｂ―10はブザーを高速でＯＮ、ＯＦＦするプログラムです。４行めではＹレジスタ（カウンタとして用いる）をクリアしています。次の５行めに関して少し述べておきましょう。解析マニュアルにもありますが、このブザー関係のリクエストでは、ＲＣＢ領域として２バイトしか必要としません。次にＢＩＯＳを呼び出してもリクエスト番号が消されたりすることはない（つまりＲＣＢ＋０は保存される）という点を考えてください。この２点から13、14行めのようなことができことがおわかりでしょうか。

13行めはブザーをＯＮにするＢＥＥＰＯＮというリクエストのＲＣＢを設定しています。これまでリクエスト番号の設定はプログラム中で、

ＬＤＡ　＃$ＯＣ

ＳＴＡ　，Ｘ

のような命令を実行することによって行うと述べてきました。それに対して，ここでは，プログラムを作る段階でＦＣＢ擬似命令を用いてリクエスト番号の設定まで行なっています。すなわち、ＲＣＢＯＮからの２バイトにはブザーをＯＮにするリクエストを呼び出すにあたって必要なリクエスト番号の設定（パラメータがあればそれもあらかじめ設定しておくこともできる）は既にプログラミングの際に終っているので、プログラム実行時にはＸレジスタにＲＣＢ領域の先頭アドレス（ここではＲＣＢＯＮ）をセットしてＢＩＯＳを呼び出すだけで良いというわけです。このようにＦＣＢ（場合によってはＦＤＢ）命令によってあらかじめリクエスト番号を設定しておくという方法はよく用いられます。しかし注意しなければいけないのは，この方法で確保したＲＣＢ領域は、汎用の(各リクエストの呼び出しで共通に使用できる)ＲＣＢ領域ではなく、設定したリクエストに対してのみ使用可能なＲＣＢ領域だという点です。ですから、この例の様にリクエスト番号が違うリクエストに対してはそれぞれについてＲＣＢ領域を確保しなければいけないので，各種のリクエストを使用する場合、ＲＣＢ領域で占める割り合いが大きくなるという欠点があります。これに対して

〔図Ｂ－10　高い音程を出す〕

```
 1                    FBFA   BIOS     EQU    $FBFA
 2   5000                             ORG    $5000
 3
 4   5000 108E 0000          START    LDY    #0
 5   5004 8E   5017          LOOP     LDX    #RCBON
 6   5007 AD   9F FBFA                JSR    [BIOS]
 7   500B 8E   5019                   LDX    #RCBOFF
 8   500E AD   9F FBFA                JSR    [BIOS]
 9   5012 31   3F                     LEAY   -1,Y
10   5014 26   EE                     BNE    LOOP
11   5016 39                          RTS
12
13   5017 0C 00              RCBON    FCB    $0C,0
14   5019 0D 00              RCBOFF   FCB    $0D,0
15
16                                    END    START

0 ERROR(S) DETECTED

SYMBOL TABLE:

BIOS   FBFA   LOOP   5004   RCBOFF 5019   RCBON  5017   START  5000
```

汎用のＲＣＢ領域を用いてプログラム実行時にリクエスト番号などをセットする方法では、プログラムの実行部は若干長くなるものの、ＲＣＢ領域は最大で８バイトしか必要ありません。

結局５～８行めまでは、ブザーをＯＮにしてからすぐにＯＦＦにする動作をしていることになります。

９行めはカウンタとしてもちいているＹレジスタをデクリメント（Ｙ←Ｙ－１）しています。

そして10行めで、デクリメントした結果が０でなければ再びＬＯＯＰへ戻る構造です。

ですからこのプログラムは、ブザーのＯＮ、ＯＦＦを即座に＄１００００＝65536回行うプログラムです。

このプログラムを実行すると、いつもとは違って周波数の高い（音程の高い）音が聞こえるはずです。これは高速でブザーをＯＮ・ＯＦＦすることによる効果で、ＦＭ－８でのゲームなどでよく用いられている方法です（ＦＭ－８にはＰＳＧがありません）。

それでは、もう一段階発展させてより効果音らしくすることを考えてみましょう。ブザーをＯＮ・ＯＦＦする際ＯＮ・ＯＦＦの間隔をかえると、それに伴って周波数（音程）もかわります。そこで、間隔を徐々に小さくしていくことよって、尻あがりな音を出すことを実現してみます。

図Ｂ－11がそのプログラムです。ＢＩＯＳの使用法は既に述べた図Ｂ－10と同じですから、ここではループの作り方に重点を置いて解説します。

通常ループを形成するには、３つの部分を作成しなければなりません。その３つとは①ループカウンタを初期化する部分、②カウンタを変化させる部分、③カウンタの値によってループするかを判断する部分です。例として100回のループを考えるとすると、Ｂレジスタをカウンタとして用いる

〔図Ｂ－11　効果音ルーチン〕

```
 1                FBFA   BIOS    EQU    $FBFA
 2   5000                        ORG    $5000
 3
 4   5000 C6   0A        START   LDB    #10 ………………………… 10回繰り返す
 5   5002 34   04                PSHS   B
 6   5004 C6   FF        LOOP0   LDB    #$FF ……………… 255段階に音程を変化させる
 7   5006 34   04                PSHS   B
 8   5008 8E   502C      LOOP1   LDX    #RCBON  ┐
 9   500B AD   9F FBFA           JSR    [BIOS]  ┘……………… BEEPON
10   500F E6   E4                LDB    ,S      ┐
11   5011 5A             LOOPON  DECB           │……………… 時間待ち（ONの時間）
12   5012 26   FD                BNE    LOOPON  ┘
13   5014 8E   502E              LDX    #RCBOFF ┐
14   5017 AD   9F FBFA           JSR    [BIOS]  ┘……………… BEEPOFF
15   501B E6   E4                LDB    ,S      ┐
16   501D 5A             LOOPOF  DECB           │……………… 時間待ち（OFFの時間）
17   501E 26   FD                BNE    LOOPOF  ┘
18   5020 6A   E4                DEC    ,S
19   5022 26   E4                BNE    LOOP1 ………………………… 音程のループ
20   5024 35   04                PULS   B
21   5026 6A   E4                DEC    ,S
22   5028 26   DA                BNE    LOOP0 ………………………… 回数のループ
23   502A 35   84                PULS   B,PC
24
25   502C 0C 00          RCBON   FCB    $0C,0
26   502E 0D 00          RCBOFF  FCB    $0D,0
27
28                               END    START

0 ERROR(S) DETECTED

SYMBOL TABLE:

BIOS    FBFA    LOOP0   5004    LOOP1   5008    LOOPOF 501D    LOOPON 5011
RCBOFF  502E    RCBON   502C    START   5000
```

とすれは、図Ｂ—12のようにするのが一般的です。もちろん考え方をかえて図Ｂ—13のようにカウンタを増加させるという方法もありますが、カウンタが増加していなければならない場合を除いてただ指定回数ループすれば良いと言う場合には図Ｂ—12の方法の方がよく用いられます。図Ｂ—12の方式では、多くの場合②はループ内処理の最後におかれるので、②と③の部分は

ＤＥＣＢ

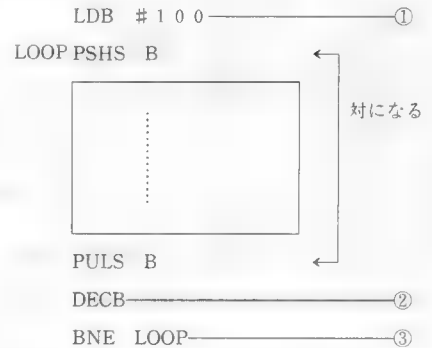
ＢＮＥ ＬＯＯＰ

とすることができ、図Ｂ—13の方式に比べて実行速度も早くて便利です。

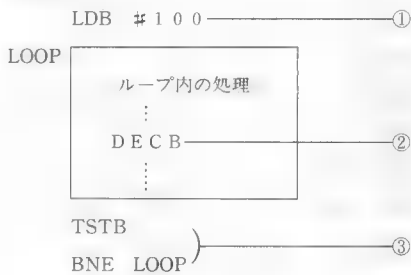
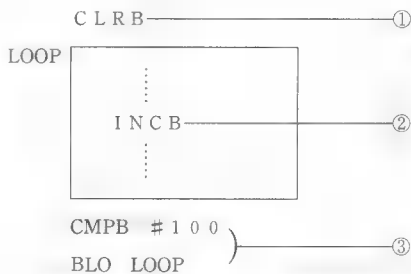
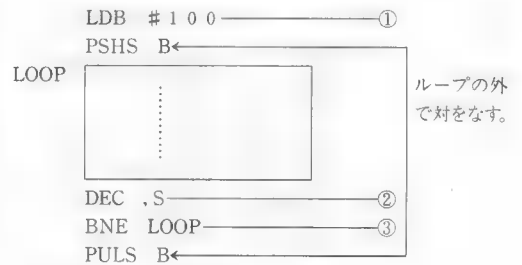
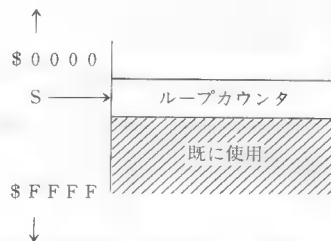
さて図Ｂ—12のループですが，ループ内の処理においてＢレジスタをどうしても使わなければいけないという場合にはどうしたらよいでしょうか。こういった問題が生じた場合の定石は、スタックに退避してしまうという方法です。すなわち図Ｂ—14のようにしてしまうわけです。こうすればループ内の処理でＢレジスタを破壊してもだいじょうぶです。しかし、ひとつだけ気になる点があります。ループというのは、ループ内の処理を多くの回数実行させるためのものです。となると、数多く実行されるループ内の処理に時間のかかるスタック操作（スタックへの退避、スタックからの復帰）をおくというのは、実行速度に大きく影響するわけです。

そこで、スタック操作をループの外に追い出してみたのが図Ｂ—15です。まずＢレジスタに初期値を設定した後Ｂレジスタをスタックに退避してしまいます。そして、②の部分は

ＤＥＣ ，Ｓ

なる命令で実現します。これはループの外でプッシュしたカウンタがスタックの一番上（図参照）

図Ｂ－14 ループの構造(C)

図Ｂ－12 ループの構造(A)

CＬＲＢ ①
LOOP
ＩＮＣＢ ②
CMPB ＃１００
BLO LOOP ③

図Ｂ－13 ループの構造(B)

ループ内でのスタックの状態

図Ｂ－15 ループの構造(D)

にあるので、それをデクリメントしています。こうすれば、図B―14の場合に比べてより速く短いループを構成することができます。ただし注意しなければいけないのは,ループを脱出したとき(BNEで分岐しなかった場合）にきちんと、スタック上のカウンタを復帰しておかなければいけないことです。これをしないと、即暴走につながりますから注意してください。

さて、図B―11のプログラムは，このループのテクニックを使用して作られています。さらに時間待ちの部分では，スタック上のカウンタ(内側)の値を利用して時間待ちの長さを決定しています。このプログラムでは「キューイ」といった感じの音が出ますが、6行めの値をかえたり、プログラムを変えたりして、自分なりの音を実現してみるのも良いでしょう。

〔図B－16　ゲームのタイトル表示〕

```
100 PRINT CHR$(12)
110 FOR I=3 TO 1 STEP -1
120 SYMBOL (160+I,I),"ZIG ZAP",5,3,1
130 NEXT I
140 SYMBOL (160,0),"ZIG ZAP",5,3,5
150 LOCATE 20, 4 :PRINT "===== PART I  / Version X.XX ====="
160 LOCATE  8, 6 :PRINT "ｺﾉ ｹﾞｰﾑ ﾊ ｢FM-7/FM-NEW7/FM-77 ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏﾆｭｱﾙ｣ ﾉ ﾀﾒﾉ ﾌﾟﾛｸﾞﾗﾑ ﾃﾞｽ｡"
170 LOCATE 20, 8 :PRINT "<< ｱ ｿ ﾋﾞ ｶ ﾀ >>"
180 LOCATE 20,10 :PRINT " ｳｴｶﾗ ｵﾘﾃｸﾙ ｴｲﾘｱﾝ ｦ ｷｬﾉﾝ ﾉ ﾊｶｲ ｺｳｾﾝ ﾃﾞ ｹﾞｷﾊ ｼﾃ ｸﾀﾞｻｲ｡"
190 LOCATE 20,12 :PRINT "<< ｷ - ﾉ ｿ ｳ ｻ >>"
200 LOCATE 18,13 :PRINT "     ┌─┐                        ┌─┐    "
210 LOCATE 18,14 :PRINT "<-- │4│ LEFT MOVE  /  RIGHT MOVE │6│ -->"
220 LOCATE 18,15 :PRINT "     └─┘                        └─┘    "
230 LOCATE 18,16 :PRINT "   ┌─────┐     ┌───────┐              "
240 LOCATE 18,17 :PRINT "   │BREAK│ FIRE │ SPACE │ GAME START "
250 LOCATE 18,18 :PRINT "   └─────┘     └───────┘              "
260 LOCATE 20,20 :PRINT "COPY RIGHT (C) 1984 by H.NAKAMURA"
```

〔図B－17　文字列の出力〕

```
 1                       FBFA  BIOS    EQU   $FBFA
 2   5000                              ORG   $5000
 3   5000 8E    502D           START   LDX   #RCB ........RCB領域先頭アドレス
 4   5003 86    14                     LDA   #$14 ........リクエスト番号
 5   5005 A7    84                     STA   ,X
 6   5007 108E  5018                   LDY   #STRING .....文字列の先頭アドレス
 7   500B 10AF  02                     STY   2,X
 8   500E CC    0015                   LDD   #21 .........文字数
 9   5011 ED    04                     STD   4,X
10   5013 AD    9F FBFA                JSR   [BIOS]
11   5017 39                           RTS   ......文字数は2バイトで表わすのでDレジスタを使う
12
13   5018 46 4D 2D 37      STRING      FCC   /FM-7ｼﾘｰｽﾞ = ｺｳﾌﾝﾊﾟｿｺﾝ/
     501C BC D8 B0 BD
     5020 DE 20 3D 20
     5024 BA B3 CC DD
     5028 CA DF BF BA
     502C DD
14
15   502D                  RCB         RMB   6 ...........RCB領域
16
17                                     END   START

0 ERROR(S) DETECTED

SYMBOL TABLE:

BIOS   FBFA   RCB    502D   START  5000   STRING 5018
```

5. 文字列出力の実践

前節では最も簡単なリクエスト（パラメータおよび復帰情報なし）を扱ってみましたが、この節ではパラメータがあるリクエストの使い方を解説します。

図B－16をみてください。これは＄D章で作成するゲームのタイトル表示部をＢＡＳＩＣで書いたものです。ここでは文字列出力(ＯＵＴＰＵＴ)のリクエストを使用して、このＢＡＳＩＣプログラムの100､150～260行をマシン語のプログラムに変換することを考えていきます。この過程をハンドコンパイル（手でコンパイルする）といいます。

まずは，その準備として（現在のカーソル位置に)“FM－７シリーズ　＝　コウフンパ゜ソコン”と表示させることを考えます。まず、ＯＵＴＰＵＴというリクエストのリクエスト番号とパラメータを解析マニュアルまたはシステム仕様、本書付録で調べます。すると、リクエスト番号は＄１４、パラメータはＲＣＢ＋２，３に出力するデータの先頭アドレス、ＲＣＢ＋４，５に出力するデータの長さ（バイト数）をセットすれば良いことがわかります。

これだけわかればもうすぐにプログラムが作成できます。図B－17がそのプログラム例です。６～７行めで出力したい文字列の先頭アドレスをＲＣＢ＋２、ＲＣＢ＋３にセットし、８～９行めで出力したい文字列の長さ（文字数）をＲＣＢ＋４、ＲＣＢ＋５にセットしています。このプログラムを実行してみた様子が図B－18です。さらに、図B－17のプログラムをポジションインディペンデ

〔図B－18　実行例〕

```
LOADM "S-B-16 ............................オブジェクト・プログラム＝マシン語プログラムをロード
Ready
EXEC &H5000 ..................................................＄５０００番地から実行
FM-7ｼﾘｰｽﾞ  = ｺｳﾌﾝﾊﾟｿｺﾝ .........................................出力された文字列
Ready
```

〔図B－19　図B－17のポンションインディペンデント版〕

```
 1                  FBFA  BIOS    EQU   $FBFA
 2   5000                         ORG   $5000
 3   5000 30   8D 002A    START   LEAX  RCB,PCR
 4   5004 86   14                 LDA   #$14
 5   5006 A7   84                 STA   ,X
 6   5008 31   8D 000D            LEAY  STRING,PCR
 7   500C 10AF 02                 STY   2,X
 8   500F CC   0015               LDD   #21
 9   5012 ED   04                 STD   4,X
10   5014 AD   9F FBFA            JSR   [BIOS]
11   5018 39                      RTS
12
13   5019 46 4D 2D 37     STRING  FCC   /FM-7ｼﾘｰｽﾞ = ｺｳﾌﾝﾊﾟｿｺﾝ/
14
15   502E                 RCB     RMB   6
16
17                                END   START

0 ERROR(S) DETECTED

SYMBOL TABLE:

BIOS    FBFA    RCB     502E    START   5000    STRING 5019
```

ント（位置独立。7章14節参照）にしたプログラムを図B―19にあげておきます。確かに位置独立になっているのかどうかを各自で確認してください。(13行めの左側のダンプリストは一部しか表示されていませんが、きちんとアセンブルされています。アセンブラに対してある指示を与えてやるとＦＣＣ命令などの際にダンプリストが１行で収まらない場合、２行め以降を表示しません。ここではその指定をして紙を節約しています)。

6. オーダ

さて、図B―16に戻るとやっかいなものがあることに気づくと思います。それはＬＯＣＡＴＥ文です。これを実現するＢＩＯＳのリクエストはみあたりませんし，他に方法を知りません。

といってもあわてることはありません。実は、出力する文字列にはオーダシーケンスを含むことができるので、このオーダシーケンスを用いれば、

ＰＵＴ・ＧＥＴコマンド時におけるオーダーの一覧表です。この表に出ていない＄００～＄１Ｆまでのコードは全て動作しません。

オーダー名	コード１　コード２　コード３	機能
ＥＬ	＄０５	フィールド終りまでの文字を消去
ＢＥＬ	＄０７	ベルをならします
ＢＳ	＄０８	バッファアドレスを１つ戻す
ＨＴ	＄０９	ＴＡＢ動作
ＬＦ	＄０Ａ	ラインフィールド
ＨＯＭＥ	＄０Ｂ	ホーム動作
ＥＡ	＄０Ｃ	画面クリア
ＣＲ	＄０Ｄ	復帰動作
ＳＦ	＄１１　アトリビュート	フィールド定義
ＳＢＡ	＄１２　Ｘ　Ｙ	バッファアドレス指定
ＲＣ	＄１３　文字数　コード	指定数同一文字表示
→	＄１Ｃ	カーソル右移動
←	＄１Ｄ	カーソル左移動
↑	＄１Ｅ	カーソル上移動
↓	＄１Ｆ	カーソル下移動
Lock Keyboard	＄１Ｂ　＄２３	キー入力禁止
Unlock Keyboard	＄１Ｂ　＄２２	キー入力禁止解除
Erase Key buffer	＄１Ｂ　＄３９	キーバッファ消去
Set Buffer Mode	＄１Ｂ　＄６７	キー先行入力可
Set Unbuffer Mode	＄１Ｂ　＄６８	キー先行入力禁止

注）例えば、BASICで

ＰＲＩＮＴ　ＣＨＲ＄(２７)＋"g" ↵

とすれば、先行入力が可能になります。

図B－20　オーダーシーケンス一覧表

ＬＯＣＡＴＥ文に相当する動作を実現することができます。

オーダシーケンスとはアスキーコードの＄００～＄１Ｆを使って画面表示をコントロールするもので、図Ｂ—20のような機能を持っています。この中のＳＢＡオーダを用いればＬＯＣＡＴＥ文を実現できます。すなわち、＄１２、＄０８、＄０６と順々に表示するようにしてやれば、ＬＯＣＡＴＥ　８，６と同じことをすることになります。他にも、＄０Ｃを表示すれば画面がクリアされますし、＄０７を表示すればベルがなります。(「表示する」といっても普通の文字を表示させるのと

〔図Ｂ－21　タイトルの表示マシン語版〕

```
 1                    FBFA  BIOS     EQU   $FBFA
 2                    0012  LOCATE   EQU   $12 ........ SBAオーダの最初のコードを定義
 3                    000C  CLS      EQU   $0C ........ EAオーダのコードを定義
 4  5000                             ORG   $5000
 5  5000 30     8D 0010  START       LEAX  RCB,PCR ...... RCB領域の先頭アドレスをXレジスタに
 6  5004 31     8D 0012              LEAY  STRING,PCR ... 文字列のあるアドレス
 7  5008 EC     A1                   LDD   ,Y++  ┐
 8  500A ED     04                   STD   4,X   ┘ ...... 文字列の長さをセット
 9  500C 10AF   02                   STY   2,X ........... 文字列の先頭アドレスをセット
10  500F AD     9F FBFA              JSR   [BIOS]
11  5013 39                          RTS
12
13  5014 14 00 00 00     RCB         FCB   $14,0,0,0,0,0
14
15  501A 01D0            STRING      FDB   ESTR-SSTR ..... 文字列の長さを2バイトで格納
16  501C 0C              SSTR        FCB   CLS ........... 画面クリア、
17  501D 12 14 04                    FCB   LOCATE,20,4
18  5020 3D 3D 3D 3D                 FCC   "===== PART I  / Version X.XX ====="
19  5042 12 08 06                    FCB   LOCATE,8,6
20  5045 BA C9 20 B9                 FCC   "ｺﾉ ｹﾞｰﾑ ﾊ ｢FM-7/FM-NEW7/FM-77 "
21  5063 CF BC DD BA                 FCC   "ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏﾆｭｱﾙ｣ ﾉ ﾀﾒﾉ"
22  507B 20 CC DF DB                 FCC   " ﾌﾟﾛｸﾞﾗﾑ ﾃﾞｽ｡"
23  5088 12 14 08                    FCB   LOCATE,20,8
24  508B 3C 3C 20 B1                 FCC   "<< ｱ ｿ ﾋﾞ ｶ ﾀ >>"
25  509B 12 14 0A                    FCB   LOCATE,20,10
26  509E 20 B3 B4 B6                 FCC   " ｳｴｶﾗ ｵﾘﾃｸﾙ ｴｲﾘｱﾝ ｦ ｷｬﾉﾝ ﾉ "
27  50B9 CA B6 B2 20                 FCC   "ﾊｶｲ ｺｳｾﾝ ﾃﾞ ｹﾞｷﾊ ｼﾃ ｸﾀﾞｻｲ｡"
28  50D3 12 14 0C                    FCB   LOCATE,20,12
29  50D6 3C 3C 20 B7                 FCC   "<< ｷ ｰ ﾉ ｿ ｳ ｻ >>"
30  50E7 12 12 0D                    FCB   LOCATE,18,13
31  50EA 20 20 20 20                 FCC   "     ┌─┐                 "
32  50FE 20 20 20 20                 FCC   "                    ┌─┐"
33  510E 12 12 0E                    FCB   LOCATE,18,14
34  5111 3C 2D 2D 20                 FCC   "<-- |4| LEFT MOVE  /"
35  5125 20 20 52 49                 FCC   "  RIGHT MOVE |6| -->"
36  5139 12 12 0F                    FCB   LOCATE,18,15
37  513C 20 20 20 20                 FCC   "     └─┘                 "
38  5150 20 20 20 20                 FCC   "                    └─┘"
39  5160 12 12 10                    FCB   LOCATE,18,16
40  5163 20 20 20 98                 FCC   "     ┌──────┐          ┌"
41  5177 95 95 95 95                 FCC   "──────────┐"
42  517F 12 12 11                    FCB   LOCATE,18,17
43  5182 20 20 20 96                 FCC   "    |BREAK| FIRE      |"
44  5196 20 53 50 41                 FCC   " SPACE | GAME START "
45  51AA 12 12 12                    FCB   LOCATE,18,18
46  51AD 20 20 20 9A                 FCC   "     └──────┘          └"
47  51C1 95 95 95 95                 FCC   "──────────┘"
48  51C9 12 14 14                    FCB   LOCATE,20,20
49  51CC 43 4F 50 59                 FCC   "COPYRIGHT (C) 1984 "
50  51DF 62 79 20 48                 FCC   "by H.NAKAMURA"
51                    51EC  ESTR     EQU   *
52
53                                   END   START
```

```
0 ERROR(S) DETECTED

SYMBOL TABLE:

BIOS    FBFA    CLS     000C    ESTR    51EC    LOCATE 0012     RCB     5014
SSTR    501C    START   5000    STRING  501A
```

〔図B－22　図B－21のダンプリスト〕

```
5000: 30 8D 00 10 31 8D 00 12 EC A1 ED 04 10 AF 02 AD :89  0■..1■..■｡○..ｯ.ｭ
5010: 9F FB FA 39 14 00 00 00 00 00 01 D0 0C 12 14 04 :E8  ｿ■■9.......ﾐ....
5020: 3D 3D 3D 3D 3D 20 50 41 52 54 20 49 20 20 2F 20 :80  ===== PART I  / 
5030: 56 65 72 73 69 6F 6E 20 58 2E 58 58 20 3D 3D 3D :13  Version X.XX ===
5040: 3D 3D 12 08 06 BA C9 20 B9 DE B0 D1 20 CA 20 A2 :01  ==...ｺﾉ ｹﾞｰﾑ ﾊ ｢
5050: 46 4D 2D 37 2F 46 4D 2D 4E 45 57 37 2F 46 4D 2D :F6  FM-7/FM-NEW7/FM-
5060: 37 37 20 CF BC DD BA DE 20 C6 AD B3 D3 DD 20 CF :73  77 ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏ
5070: C6 AD B1 D9 A3 20 C9 20 C0 D2 C9 20 CC DF DB B8 :62  ﾆｭｱﾙ｣ ﾉ ﾀﾒﾉ ﾌﾟﾛｸ
5080: DE D7 D1 20 C3 DE BD A1 12 14 08 3C 3C 20 B1 20 :3C  ﾞﾗﾑ ﾃﾞｽ｡...<< ｱ 
5090: BF 20 CB DE 20 B6 20 C0 20 3E 3E 12 14 0A 20 B3 :DD  ｿ ﾋﾞ ｶ ﾀ >>... ｳ
50A0: B4 B6 D7 20 B5 D8 C3 B8 D9 20 B4 B2 D8 B1 DD 20 :4E  ｴｶﾗ ｵﾘﾃｸﾙ ｴｲﾘｱﾝ 
50B0: A6 20 B7 AC C9 DD 20 C9 20 CA B6 B2 20 BA B3 BE :55  ｦ ｷｬﾉﾝ ﾉ ﾊｶｲ ｺｳｾ
50C0: DD 20 C3 DE 20 B9 DE B7 CA 20 BC C3 20 B8 C0 DE :EB  ﾝ ﾃﾞ ｹﾞｷﾊ ｼﾃ ｸﾀﾞ
50D0: BB B2 A1 12 14 0C 3C 3C 20 B7 20 B0 20 C9 20 BF :27  ｻｲ｡...<< ｷ ｰ ﾉ ｿ
50E0: 20 B3 20 BB 20 3E 3E 12 12 0D 20 20 20 20 98 95 :28   ｳ ｻ >>...    ┌─
50F0: 99 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 :79  ┐               
5100: 20 20 20 20 20 20 20 20 20 20 20 98 95 99 12 12 :4A             ┌─┐..
5110: 0E 3C 2D 2D 20 96 34 96 20 4C 45 46 54 20 4D 4F :2B  .<-- │4│ LEFT MO
5120: 56 45 20 20 2F 20 20 52 49 47 48 54 20 4D 4F 56 :DA  VE  /  RIGHT MOV
5130: 45 20 96 36 96 20 2D 2D 3E 12 12 0F 20 20 20 20 :32  E │6│ -->...    
5140: 9A 95 9B 20 20 20 20 20 20 20 20 20 20 20 20 20 :6A  └─┘             
5150: 20 20 20 20 20 20 20 20 20 20 20 20 20 9A 95 9B :6A               └─┘
5160: 12 12 10 20 20 20 98 95 95 95 95 95 99 20 20 20 :0E  ...   ┌─────┐   
5170: 20 20 20 20 20 20 98 95 95 95 95 95 95 95 99 12 :16        ┌───────┐.
5180: 12 11 20 20 20 96 42 52 45 41 4B 96 20 46 49 52 :15  ..   │BREAK│ FIR
5190: 45 20 20 20 20 96 20 53 50 41 43 45 20 96 20 47 :04  E    │ SPACE │ G
51A0: 41 4D 45 20 53 54 41 52 54 20 12 12 12 20 20 20 :37  AME START ...   
51B0: 9A 95 95 95 95 95 9B 20 20 20 20 20 20 20 20 20 :3E  └─────┘         
51C0: 9A 95 95 95 95 95 95 95 9B 12 14 14 43 4F 50 59 :BD  └───────┘...COPY
51D0: 52 49 47 48 54 20 28 43 29 20 31 39 38 34 20 62 :AA  RIGHT (C) 1984 b
51E0: 79 20 48 2E 4E 41 4B 41 4D 55 52 41 00 00 00 00 :5F  y H.NAKAMURA....
```

〔図B－23　図B－21の実行例〕

```
            ===== PART I  / Version X.XX =====

ｺﾉ ｹﾞｰﾑ ﾊ ｢FM-7/FM-NEW7/FM-77 ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏﾆｭｱﾙ｣ ﾉ ﾀﾒﾉ ﾌﾟﾛｸﾞﾗﾑ ﾃﾞｽ｡

            << ｱ ｿ ﾋﾞ ｶ ﾀ >>

             ｳｴｶﾗ ｵﾘﾃｸﾙ ｴｲﾘｱﾝ ｦ ｷｬﾉﾝ ﾉ ﾊｶｲ ｺｳｾﾝ ﾃﾞ ｹﾞｷﾊ ｼﾃ ｸﾀﾞｻｲ｡

            << ｷ ｰ ﾉ ｿ ｳ ｻ >>
                ┌─┐                                ┌─┐
           <-- │4│ LEFT MOVE  /  RIGHT MOVE │6│ -->
                └─┘                                └─┘
              ┌─────┐          ┌───────┐
              │BREAK│ FIRE     │ SPACE │ GAME START
              └─────┘          └───────┘

            COPYRIGHT (C) 1984 by H.NAKAMURA
```

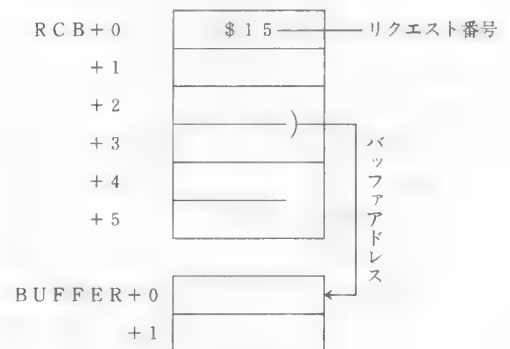
同様にするという意味で、実際には画面に文字は表示されません)。

そこで、このオーダシーケンスを用いて作成した図B―16のマシン語版が図B―21です(ここでも，FCC命令で1行めしか表示しない指定がしてあります。マシンコードは図B―22のダンプリストを参照してください)。このプログラムを実行すれば図B―23のような画面表示が得られるはずです(WIDTH　80，25を実行しておいてください)。

7. キー入力の実践

この節では前節までの実践を踏まえて、復帰情報のあるリクエストの使い方をキー入力を行うKEYINのリクエストを例にして実践してみます。

作成するプログラムは、スペースキー(アスキーコード$20)が押されるまでループし続けるプログラムです。まず図B―24をみてください。これはKEYINリクエストのRCBを示したものです。このうち、BIOSを呼び出すにあたって設定しなければいけないのは、RCB＋0のリクエスト番号と、RCB＋2、RCB＋3のバッファアドレスです。このバッファというのは、復帰情報が格納される場所のことで、RCB＋2，3にはこのバッファの先頭アドレスをセットします。

パラメータをセットしてBIOSを呼び出すとBIOSではキーボードからデータを受け取り、RCB＋2，3に格納されている値がさすアドレスと、その次のアドレスに、キーボードの情報をセットします。ここでセットされる情報は、BUFFER＋1がキー入力の有無(0ならキー入力

図B－24　KEYINのRCB

〔図B－25　KEYINリクエスト〕

```
 1                 FBFA  BIOS    EQU   $FBFA
 2   5000                        ORG   $5000        …RCB領域先頭アドレスをXレジスタにセット
 3   5000 30    8D 001A  START   LEAX  RCB,PCR
 4   5004 86    15               LDA   #$15   ┐…………リクエスト番号をセット
 5   5006 A7    84               STA   ,X     ┘
 6   5008 31    8D 001A          LEAY  BUFFER,PCR ‥Yレジスタにバッファの先頭アドレスをセット
 7   500C 10AF  02               STY   2,X ……RCB領域内にバッファの先頭アドレスをセット
 8   500F AD    9F FBFA  LOOP    JSR   [BIOS]
 9   5013 6D    21               TST   1,Y‥キー入力があったかを調べる
10   5015 27    F8               BEQ   LOOP……………ないときはLOOP
11   5017 A6    A4               LDA   ,Y‥入力されたキーのアスキーコードをAレジスタに
12   5019 81    20               CMPA  #$20‥スペースキー($20)かどうかをチェック
13   501B 26    F2               BNE   LOOP……………違ったらLOOP
14   501D 39                     RTS
15
16   501E                RCB     RMB   8……………RCB領域(8バイト分)
17   5026                BUFFER  RMB   2……………バッファ領域(2バイト分)
18
19                               END   START

 0 ERROR(S) DETECTED

 SYMBOL TABLE:

 BIOS   FBFA   BUFFER 5026   LOOP   500F   RCB   501E   START  5000
```

はなかった)、BUFFERが入力されたキーのアスキーコード(BUFFER+1=0のときはこの値は無効)となっています。

以上を踏まえて作成したのが図B―25です。このプログラムではXレジスタにRCB領域の先頭アドレスが、Yレジスタにバッファ領域の先頭アドレスがセットされています。

さて、図B―25のプログラムは正確な手順でBIOSを使用しています。そのためRCB領域はきちんと8バイト確保し、バッファ領域もRCB領域とは別に2バイト確保しています。規則どうりでいけばこうするのが正当なのですが、ちょっと細工をすると、バッファ領域の確保が不要になります。

実はこのKEYINリクエストは8バイトあるRCB領域のうち6バイトしか使用しません。すなわちRCB領域のうちRCB+6, 7の2バイトは使用しないわけです。そこでこの2バイトをバッファ領域として使用しようというわけです。こうすることによりバッファ領域としてRCB領域と別に確保する必要もなくなるわけです。これを利用して作成したのが図B―26のプログラムです。図B―25のプログラムと同じことをするのですが全体で4バイト(6行めも2バイト短くなっている)短いプログラムになっています。

8. その他のリクエスト

ここまでで、BEEPON、BEEPOF、OUTPUT、KEYINの4つのリクエストを例を用いて解説してきました。BIOSにはこの他にも20個のリクエストがありますが、本書はBIOSのリクエスト解説を目的としてはいませんので省略することにします。参考までにBIOSのリクエストを使用する機器別に分類したものを図B―27にあげておきます。

もっと各リクエストについて詳細を知りたいという方は、本体に付属してきた「ユーザーズマニュアルシステム仕様」のファームウェア仕様のところに各リクエストの説明が載っていますので読んでみると良いでしょう。

また、各リクエストの具体的な使用法を詳しく知りたい方は、拙著「FM―7　F―BASIC解析マニュアル　フェーズI　基礎編」の第1章を参照してください。各リクエストの詳しい使用法とそのリクエストを使用したサンプルプログラ

〔図B－26　KEYINリクエストPart 2〕

```
 1                     FBFA  BIOS    EQU   $FBFA
 2   5000                            ORG   $5000
 3   5000 30    8D 0018  START   LEAX  RCB,PCR
 4   5004 86    15                   LDA   #$15
 5   5006 A7    84                   STA   ,X
 6   5008 31    06                   LEAY  6,X  ……RCB領域の7バイトめと8バイトめ
 7   500A 10AF  02                   STY   2,X    (RCB+6,7)をバッファとして使
 8   500D AD    9F FBFA  LOOP    JSR   [BIOS]  用する
 9   5011 6D    21                   TST   1,Y
10   5013 27    F8                   BEQ   LOOP       ………… ここにスペースがある
11   5015 A6    A4                   LDA   ,Y
12   5017 81    20                   CMPA  #'␣         SPACE
13   5019 26    F2                   BNE   LOOP
14   501B 39                         RTS
15
16   501C                    RCB     RMB   8 …………RCB領域(8バイト)
17                                               うち2バイトをバッフ
18                                   END   START     ァとして使用する

 0 ERROR(S) DETECTED

 SYMBOL TABLE:

 BIOS   FBFA   LOOP   500D   RCB    501C   START  5000
```

ムが掲載してあります。ちなみに、当社既刊のFM—7解析マニュアルは、この基礎編、フェーズII探究編ともFM—7を対象としていますが、FM—NEW7、FM—77にもそのまま適用できます。

図B—27　BIOSリクエストの分類

第＄C章

サブシステム

――グラフィックを使う――

1. ディスプレイサブシステム

ＢＩＯＳがサポートしている装置（＄Ｂ章１節参照）の中にディスプレイサブシステムというものがあります。これは要するにサブＣＰＵのことで、その存在は既に御存知と思います。

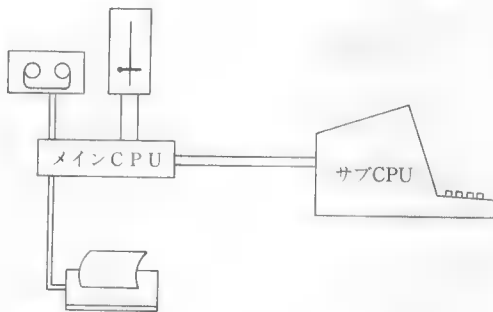
このディスプレイサブシステム（以下サブシステム）が担当している仕事は、画面処理、キー入力処理の２つです。それではまずなぜFM—7シリーズがＣＰＵを２つも使った構成を取っているかについて解説します。

FM—7シリーズ以外のパソコン（日本電気のＰＣシリーズなど）では、ＣＰＵは１つで、そのＣＰＵがＢＡＳＩＣインタプリタの実行やグラフィックの表示、キー入力の制御などのことを全て担当しています。このことは利点がないわけではないのですが、１つのＣＰＵに全ての仕事が集中するので、処理速度があがらなくなる欠点があります。その点、デュアルＣＰＵ方式（２つのＣＰＵを用いる方式のこと）では、ＣＰＵの仕事を２つのＣＰＵで分けて処理することができ、処理速度の向上が望めます（実際はＣＰＵが２つになったのだから処理速度も２倍になるというわけにはいきません）。

次にFM—7シリーズ以外のパソコンの場合、１つのＣＰＵのアドレス上にすべてのものをおかなければなりません。つまり、１つのＣＰＵのアドレス（＄００００～＄ＦＦＦＦ）の中にメインＲＡＭ、ＲＯＭ、グラフィックがあればＧＲＡＭ(グラフィックＲＡＭのこと。画面の情報を記憶しておくＲＡＭ、640×200×８色のグラフィックならば、48KバイトのＲＡＭが必要）などをすべて押し込めなければならず、いろいろな障害が生じてしまいます。この点は、ＰＣシリーズなどでもバンク切換えなどの手法を用いて改善しているのですが、やはり荷が重いというのは事実です。この点デュアルＣＰＵ方式では各ＣＰＵにそれぞれに必要なものだけを割り当てればよいので負担が軽くなります。

以上の利点があるわけですが、一方では欠点も伴っています。例えば、デュアルＣＰＵ方式の場

合には処理が分散するために、２つのＣＰＵの間で、きちんとした連携が取れていなければならず、その点を考えてプログラミングをしなければいけません。また、デュアルＣＰＵを完全に使いこなそうとすれば、それぞれのＣＰＵに対してプログラミングをしなければならず繁雑となります。

さて、このような背景を持ったディスプレイサブシステムですが､簡単にみれば図Ｃ－2にあるようにカセットテープやディスク、プリンタなどと同様につながれた、端末装置（キーボードとディスプレイがついていて、コンピュータに人間が指令を伝える操作卓のようなもの。ターミナルともいい、大型コンピュータでよく使われる）の一種とみなすことができます。

図C－2　サブCPUのイメージ

2. サブシステムの使い方Ｉ

それでは、サブシステムの具体的な使用法を解説していきましょう。

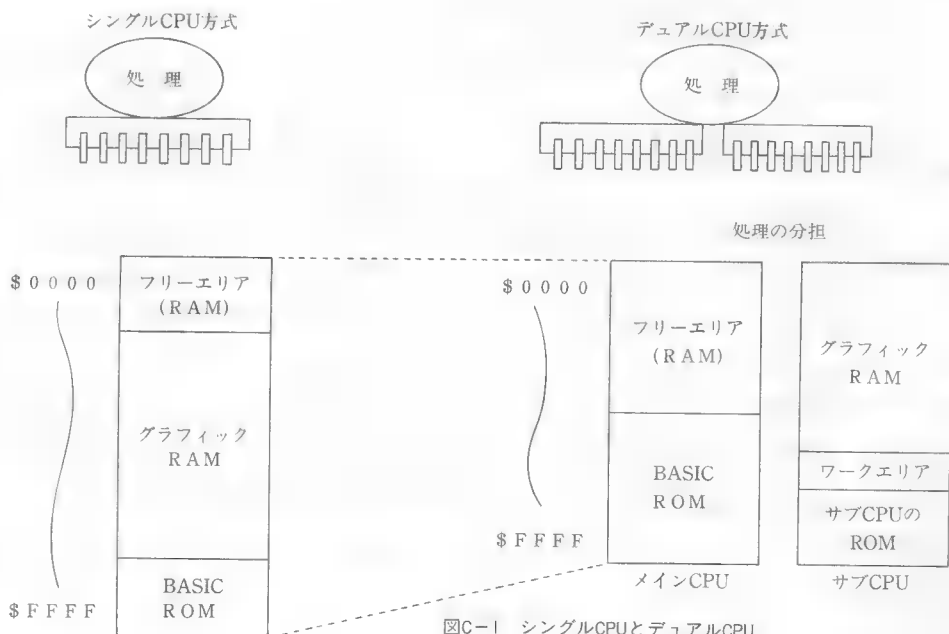
サブＣＰＵすなわちサブシステムを動作させるには、ＢＩＯＳの場合と同じようにサブシステムに対して各種のコマンドを送ってやる必要があります。このコマンドを**サブシステムコマンド**といいます。ＢＩＯＳの場合と同様に、サブシステムにコマンドを送るためには一定の手順を踏んで行わなければいけません。低レベルな(本質に近い)みかたをすれば、各種信号線を操作してサブＣＰＵをコントロールしコマンドを与えることをしなければいけません。この手順は複雑で手間を要するやっかいな仕事です。

この手間を省くためのものがＢＩＯＳのＳＵＢＯＵＴ、ＳＵＢＩＮリクエストです。また、文字列の出力やキー入力など良く使われるサブシステムコマンドを用いる場合には、ＢＩＯＳ内のさらに専用化されたリクエスト（ＩＮＰＵＴ、ＯＵＴ

処理の分担

$００００
フリーエリア (ＲＡＭ)
グラフィック ＲＡＭ
BASIC ＲＯＭ
$ＦＦＦＦ

$００００
フリーエリア (ＲＡＭ)
BASIC ＲＯＭ
$ＦＦＦＦ
メインCPU

グラフィック ＲＡＭ
ワークエリア
サブCPUの ROM
サブCPU

図C－１　シングルCPUとデュアルCPU

コード16進	コマンド名	内　　容	BIOS(注)
0 1	INIT	コンソール初期化	○
0 2	ERASE	画面クリア（文字色指定なし）	○
0 3	PUT	文字列表示	○
0 4	GET	文字列表示・入力	×
0 5	GETC	GETの未転送分を転送	×
0 6	GCBLK 1	枠内文字コード読み取り	×
0 7	PCBLK 1	枠内文字コード表示	○
0 8	GCBLK 2	枠内文字コードおよび属性読取り	×
0 9	PCBLK 2	枠内文字コードおよび属性表示	○
0 A	GBADR	バッファアドレス読取り	×
0 B	TABSET	TAB位置設定	○
0 C	CNSCTL	コンソール機能設定	○
0 D	ERASE 2	画面クリア（文字色指定あり）	○
1 5	LINE	直線又は四角の表示	○
1 6	CHAIN	連続直線表示	○
1 7	POINT	点表示	○
1 8	PAINT	ペイント	○
1 9	SYMBOL	文字列表示（大きさ・角度付き）	○
1 A	CCOLOR	枠内の色を変更	○
1 B	GGBLK 1	枠内ドット読取り	×
1 C	PGBLK 1	枠内ドット表示	○
1 D	GGBLK 2	枠内ドット読取り（色付き）	×
1 E	PGBLK 2	枠内ドット表示（色付き）	○
1 F	GCURS	座標読み取り	×
2 0	CLINE	文字による直線または四角の表示	○
2 9	INKEY	キーコード読取り	×
2 A	DEFPF	PFキーに文字列を安義	○
2 B	GETPF	PFキー定義文字列読み取り	×
2 C	INTCTL	PFキー割込み選択	○
3 D	SETIME	タイマセット	○
3 E	GETIME	タイマ読取り	×
3 F	TEST	メンテナンスコマンド	**
6 4	CONT	未転送データ転送	*

注）BIOSの項は、BIOSのSUBOUTで使用することができるか否かを示してあります。
　○：SUBOUT、SUBINどちらも可。
　×：SUBINのみ可。
　*：このコマンドを使用する前のコマンドによる。
　**：メンテナンスコマンドの用途による。

サブシステムコマンド一覧表

エラーコード（16進）	エラー内容
3 C	INITコマンドパラメータエラー
3 D	コンソール座標エラー
3 E	オーダーエラー
3 F	グラフィック座標エラー
4 0	ファンクションコードエラー
4 1	座標数エラー
4 2	文字数エラー
4 3	色数エラー
4 4	PF番号エラー
4 5	コマンドパラメータエラー
4 6	コマンドコードエラー

注）このエラーコードはBIOS経由で使用したときには、そのままBIOSのエラーコードとなります。

サブシステムエラーコード一覧表

図C－3

PUT、KEYINなど）を用いれば、ほとんどサブシステムを意識せずにおくことができます。

そこでこの節では、BIOSを用いて手間を省いたサブシステムの使用法を解説していきますが、まずは、サブシステムコマンドにどのようなものがあるのかをみていきましょう。図C―3がサブシステムコマンドの一覧です。

大まかに、文字表示関係、グラフィック関係、キーボード関係、タイマ関係、その他の5つにわけられます。この中からSYMBOLというコマンドを例にあげて解説します。

図C－4

0	$00	
1	$00	
2	$19	コマンドコード
3	CL	カラーコード(0～7)
4	F	ファンクションコード(0～5)
5	A	アングルコード(下図参照, 0～3)
6	W	文字横幅倍率(0～255)
7	H	文字縦幅倍率(0～255)
8–9	X	X座標(0～639)
10–11	Y	Y座標(0～199)
12	N	文字数(1～80)
13	String	文字列
～		
N+12		

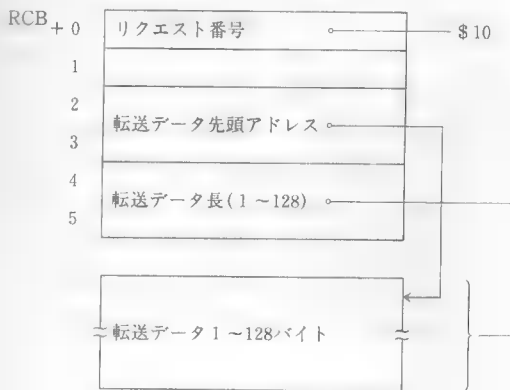
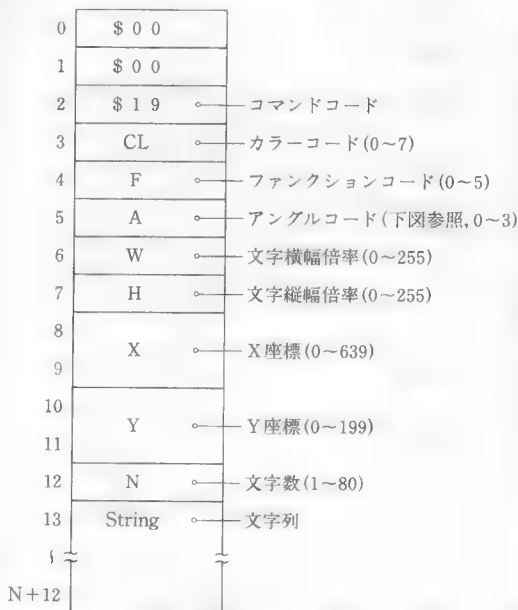
図C－5

まずBIOSのSUBOUTリクエストを参照してください（図C―4)。**転送データ**というのはサブシステムコマンドのことですから（転送データというわけは後ほど）その部分は各サブシステムコマンドの解説（システム仕様または解析マニュアル）を参照します。ここではSYMBOLというサブシステムコマンドを用いるので、そこを参照すると図C―5のようになっています。左側に示されている数字は、**転送データ先頭アドレスからの相対値**です。

それぞれの内容を少し解説すると、相対値2は**コマンドコード**と呼ばれるものです。このコードによってサブシステムが行う仕事の種類が選択されるもので、BIOSでのリクエスト番号と同じようなものです。このコマンドコードはどんなサブシステムコマンドでも相対値2のところに置くことになっています。相対値3は色を指定する**カラーコード**です。相対値4は**ファンクションコード**と呼ばれるものです。これは、BASICのグラフィック関係のコマンドで用いる、PSET、PRESETなどに対応するもので、図C―6のように数が決まっています。相対値5は図C―5の下に書いてあるように文字表示の向きを指定するものです。相対値6、7は、文字の拡大率、相対値8～11は表示する座標です。相対値12は表示しようとする文字の文字数を示し、相対値13～は文字列をセットします。BASICでSYMBOL文を用いる場合には、一部のパラメータを省略することができましたが、この場合はそういうことは許されずすべてのパラメータを設定する必要があります。各パラメータの役割は今述べたとおりですが、BASICのSYMBOL文を対応させるならば、

SYMBOL (X, Y), "String", W, H, CL, A, F

という感じになります。

結局、設定しなければいけないのは、サブシステムコマンドのコマンドコードとそれに伴うパラメータ、BIOSのリクエストコードとそれに伴うパラメータということになります。これらを設

定してＢＩＯＳを呼べば、ＢＩＯＳが決められた手順にそってサブシステムにサブシステムコマンドを実行させてくれます。

では実際のプログラムで見てみましょう。図Ｃ—7は、図Ｂ—16のＢＡＳＩＣプログラムの140行に相当するプログラムです。

4～10行めは、ＢＩＯＳに対するパラメータの設定を行っている部分です。7行めは、サブシステムコマンド（すなわち転送データ）の先頭アドレス、9行めはサブシステムコマンドの長さを設定する部分です。そして14行めはＲＣＢ領域です。

15～20行めがサブシステムコマンドの設定部分です。ここではFCB 命令などを用いてあらかじめプログラム中に設定しておく方法を取っています。この部分はＢＩＯＳの部分と同様、各パラメータごとに例えば（Ｙレジスタにサブシステムコマンドの先頭アドレスが入っているとして）

```
LDA #$19
STA 2,Y
```

のようにやっていってもよいのですが、サブシステムコマンドの場合、ＢＩＯＳと違って設定しなければならないパラメータが多いので、この方式が用いられることが多いようです。

コード	機能名	機能
0	PSET	指定色のドットを表示
1	PRESET	ドットを背景色にする
2	OR	出力色と画面色のORを取って表示
3	AND	出力色と画面色のANDを取って表示
4	XOR	出力色と画面色のXORを取って表示
5	NOT	ドットの補色をとる

図C−6　ファンクション・コード表

〔図Ｃ－７　ＳＹＭＢＯＬコマンド〕

```
    1                    FBFA  BIOS     EQU    $FBFA
    2   5000                            ORG    $5000
    3
    4   5000 8E    5018        START    LDX    #RCB
    5   5003 86    10                   LDA    #$10
    6   5005 A7    84                   STA    ,X
    7   5007 108E  5020                 LDY    #SUBCOM
    8   500B 10AF  02                   STY    2,X
    9   500E CC    0014                 LDD    #ENDCOM-SUBCOM
   10   5011 ED    04                   STD    4,X
   11   5013 AD    9F FBFA              JSR    [BIOS]
   12   5017 39                         RTS
   13
   14   5018                   RCB      RMB    8
   15   5020 00 00 19 05       SUBCOM   FCB    0,0,$19,5
   16   5024 00 00 05 03                FCB    0,0,5,3
   17   5028 00A0 0000                  FDB    160,0
   18   502C 07                         FCB    7
   19   502D 5A 49 47 20                FCC    /ZIG ZAP/
        5031 5A 41 50
   20                   5034   ENDCOM   EQU    *
   21

 0 ERROR(S) DETECTED

 SYMBOL TABLE:

 BIOS    FBFA    ENDCOM 5034    RCB     5018    START   5000    SUBCOM 5020
```

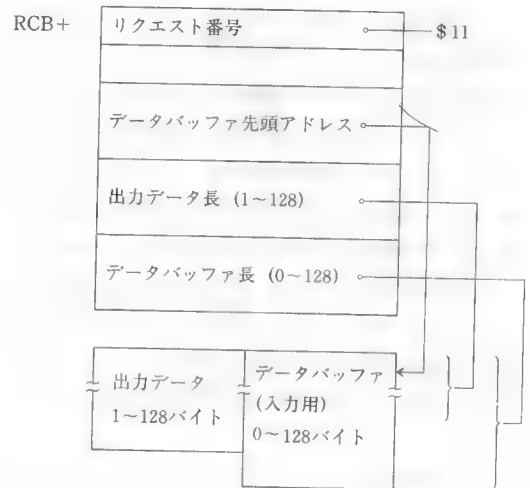
このプログラムを実行すると画面上部にシアン（水色）で“ZIG　ZAP”（第$D章で作成するゲームの名前）と表示します。

以上でSUBOUTリクエストを用いてのサブシステムの使用法はわかったと思います。メインCPUからの指令が単一方向の場合、つまり、サブシステムへコマンドを与えて仕事をさせるだけで、サブシステムからデータを返してもらうことを期待しない場合（文字列やグラフィック表示などのようにやりっぱなしでよい場合）には、SUBOUTリクエストを用いることでサブシステムを動かすことができます。

ところが、サブシステムからデータを返してもらうことを期待する場合（キー入力やタイマの読取りなどの場合）には、SUBOUTリクエストのようにやりっぱなしではデータを返してもらうことができません。そこでデータを返してもらうことを期待する場合にはSUBINリクエスト（リクエスト番号$11）を用いることになっています。

それではタイマを読み取るプログラム（図C—8）を例に解説しましょう。

まず図C—9をみてください。これがSUBOUTリクエストにおけるRCB領域の構成です。デ

図C－9

〔図C－8　タイマ読み取り〕

```
 1                 FBFA  BIOS    EQU   $FBFA
 2   5000                        ORG   $5000
 3
 4   5000 8E   5025      START   LDX   #RCB·····················R C B領域先頭アドレス
 5   5003 86   11                LDA   #$11 ┐
 6   5005 A7   84                STA   ,X   ┘····················S U B I Nリクエスト
 7   5007 108E 5100              LDY   #SUBCOM··················データバッファ先頭アドレス
 8   500B 6F   A4                CLR   ,Y   ┐
 9   500D 6F   21                CLR   1,Y  ┘····················相対値0，1をクリア
10   500F 86   3E                LDA   #$3E·····················コマンドコード
11   5011 A7   22                STA   2,Y
12   5013 10AF 02                STY   2,X ····データバッファ先頭アドレスをR C B領域にセット
13   5016 CC   0003              LDD   #3   ┐
14   5019 ED   04                STD   4,X  ┘····················出力データ長をセット
15   501B CC   0015              LDD   #21  ┐
16   501E ED   06                STD   6,X  ┘····················入力データ長をセット
17   5020 AD   9F FBFA           JSR   [BIOS]···················B I O Sコール
18   5024 39                     RTS
19
20   5025                RCB     RMB   8························R C B領域
21
22   5100                        ORG   $5100
23   5100                SUBCOM  RMB   21·······················データバッファ領域
24
25                               END   START

0 ERROR(S) DETECTED

SYMBOL TABLE:

BIOS    FBFA   RCB    5025   START  5000    SUBCOM 5100
```

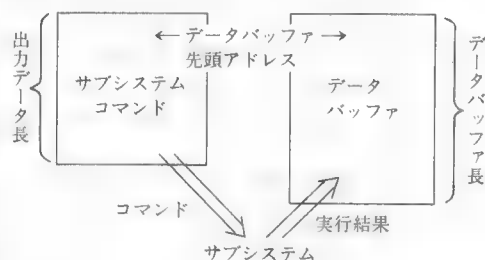
ータバッファ先頭アドレスというのは、サブシステムとのデータのやりとりに使用するデータバッファ領域の先頭アドレスです。これはＳＵＢＩＮリクエストにおける転送データ先頭アドレス（＝サブシステムコマンド先頭アドレス）とほぼ同じです。出力データ長というのは、データバッファに設定されている、サブシステムへ与えるデータ＝サブシステムコマンドの長さを示しています。ＢＩＯＳはサブシステムに、ここに示したバイト数だけデータを送ります。データバッファ長というのは、サブシステムから受け取るデータ＝実行結果やエラー番号（サブシステムがエラーをおこすこともあります）をセットする領域の長さを示しています。ＢＩＯＳは、サブシステムの実行終了後、実行結果やエラー番号をデータバッファ先頭アドレスからここに示すバイト数だけ順々にセットします。

ここで注意しなければならないのは、データバッファは、出力（＝サブシステムコマンド）と入力（＝実行結果他）について共通だということです。ですから（データバッファ長に0を指定したのでない限り）出力としてセットしたサブシステムコマンドは破壊されることになります。そのため、前述のＳＵＢＯＵＴコマンドの例と同じくサブシステムコマンドのセットをFCBなどの定数設定の擬似命令で行うことはできません（もちろん、データバッファ長に0を設定した場合や破壊されても構わないのであれば別です)。きちんと、次のようにサブシステムコマンドを設定しなければなりません。

```
LDA  #$3E
STA  2,Y
```

さらに、このＳＵＢＩＮリクエストの場合、ＢＩＯＳのリクエストとしても復帰情報があり、ＲＣＢ＋4，5に、サブシステムコマンド実行後、データバッファに実際にセットされたデータの長さ（通常ＲＣＢ＋6，7のデータバッファ長と同じになります）がセットされます。ですから、ＲＣＢ領域に関しても、FCBなどの定数設定の擬似命令は使用できないことになります。ただし実際はＲＣＢ＋4，5が破壊されるだけですからこの部分さえ再設定するようにプログラムを組めばFCB命令などを用いることはできます。

次に、タイマ読取り（ＧＥＴＩＭＥ）のサブシステムコマンドを図Ｃ—11にあげました。実行前にセットしなければならないのは、相対値（サブシステムコマンド先頭アドレスに対する相対値）0～2までです。このうち相対値0と1への0のセットは、通常の場合省略しても構いません（特殊な処理の後などに必要になるものです)。そして、返される情報は、全部で21バイトになります（これらの詳しい内容についてはシステム仕様または解析マニュアルを参照のこと)。

セットしなければならない事項がわかれば図Ｃ—8のプログラムの理解は容易でしょう。ＲＣＢ領域、データバッファ領域ともRMBで領域を確保している点に注意してください。図Ｃ—12がこのプログラムを実行した様子です。プログラムの実行後に、ＲＣＢ領域が設定され、サブシステムからデータが送られてきていることに注意してください。

図C－10 データバッファ

図C－11 GETIMEコマンド

3. サブシステムの使い方II

前節では、ＢＩＯＳを使用してサブシステムを制御し、サブシステムコマンドを実行させる方法を述べました。ＢＩＯＳを使用する場合には、細かい操作から解放されるという利点があります。しかし一方で、ＢＩＯＳを使用すると、実行速度が落ちてしまうという欠点もあります。また、ＢＩＯＳを使用することができない場合（裏ＲＡＭを使用するときには、ROM上のＢＩＯＳを使用することはできない）もあります。

そこでこの節ではＢＩＯＳを使用しないでサブシステムを制御する方法について、その手順にそった形で解説していきます。

まず、サブＣＰＵとメインＣＰＵとの関係を見てみましょう。この２つのＣＰＵは、必要に応じて、お互いに連絡を取りあっていなければいけません。そのため、サブＣＰＵとメインＣＰＵはそれぞれのメモリの一部を共有して、そこを連絡場所として使用することによって、連絡（すなわちコマンドやパラメータ、結果などのやり取り）をとっています。この共有しているメモリのことを共有ＲＡＭといいます。この共有ＲＡＭは、メイ

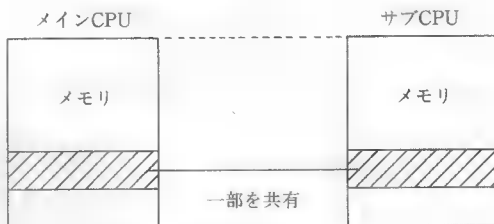
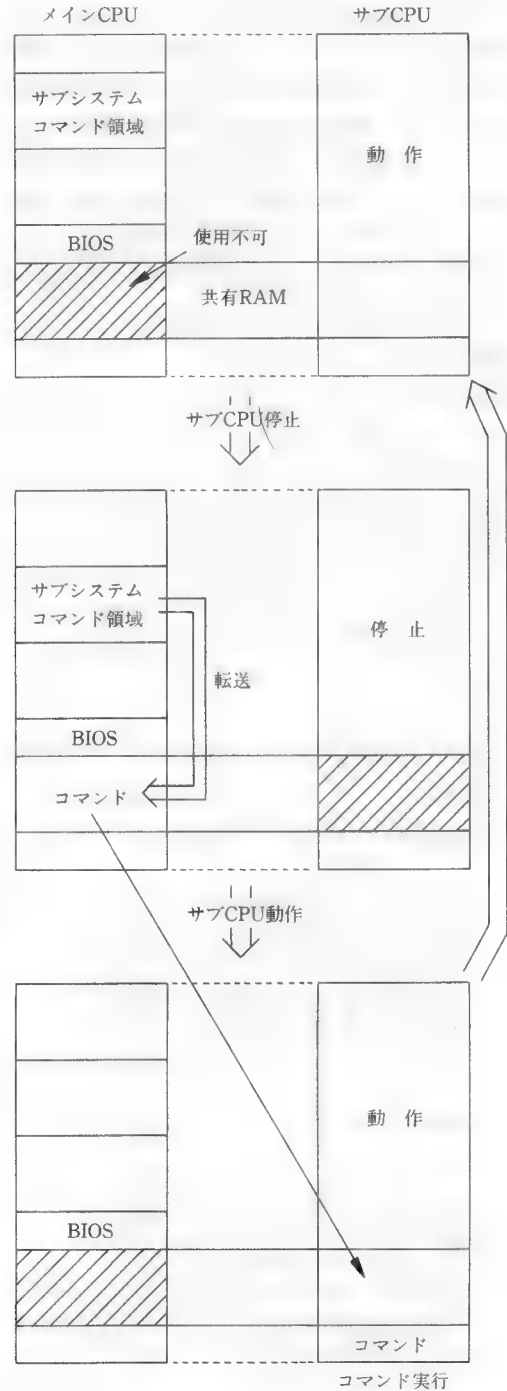
〔図Ｃ－12　図Ｃ－８の実行例〕

```
*D5000

5000 8E 50 25 86 11 A7 84 10
5008 8E 51 00 6F A4 6F 21 86
5010 3E A7 22 10 AF 02 CC 00
5018 03 ED 04 CC 00 15 ED 06
5020 AD 9F FB FA 39 00 00 00
5028 00 00 00 00 00 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*

EXEC &H5000

Ready
MON

*D5000

5000 8E 50 25 86 11 A7 84 10
5008 8E 51 00 6F A4 6F 21 86
5010 3E A7 22 10 AF 02 CC 00
5018 03 ED 04 CC 00 15 ED 06
5020 AD 9F FB FA 39 11 00 51
5028 00 00 15 00 15 00 00 00
5030 00 00 00 00 00 00 00 00
5038 00 00 00 00 00 00 00 00
*

Break

*D5100

5100 00 00 00 00 08 15 3A 05
5108 05 00 00 00 00 FF FF AC
5110 85 00 00 00 00 00 00 00
5118 00 00 00 00 00 00 00 00
5120 00 00 00 00 00 00 00 00
5128 00 00 00 00 00 00 00 00
5130 00 00 00 00 00 00 00 00
5138 00 00 00 00 00 00 00 00
*
```

ン側には、＄ＦＣ８０～＄ＦＣＦＦの128バイトに接続されています。

ＢＩＯＳのＳＵＢＯＵＴやＳＵＢＩＮのリクエストでは、与えられたサブシステムコマンドを、サブシステムコマンド領域から、この共有ＲＡＭへ転送して、サブＣＰＵ＝サブシステムに仕事をさせ、得られた結果（ＳＵＢＩＮの場合）をこの共有ＲＡＭから、データバッファ領域へと転送するというわけです。

さて、この共有ＲＡＭですが、サブＣＰＵとメインＣＰＵが同時に読み取ったり書き込んだりしたらどうなるでしょうか。こうなると、メインＣＰＵが書き込んだつもりが、サブＣＰＵが別の値を書き込んでいたなどということなどが生じてしまいます。そこで、この共有ＲＡＭは、サブＣＰＵとメインＣＰＵが同時に使用したりすることがないようになっています。通常は共有ＲＡＭはサブＣＰＵ側に接続されていて、メインＣＰＵは使用できなくなっています。そして、必要に応じて、メインＣＰＵ側に接続し、その時にはサブＣＰＵが使用できないようにするというシステムになっています。

このあたりがどうなっているかをもう少し詳しくみると次のようになります。通常は共有ＲＡＭはサブＣＰＵに接続されており、サブＣＰＵはそこに設定してあるコマンドを自由に読み取り処理を行うことができます。この時には、メインＣＰＵは共有ＲＡＭの内容をみることや書き込むことはできない仕掛けになっています。次に、メインＣＰＵがサブシステム＝サブＣＰＵに仕事をしてほしくなったときには、共有ＲＡＭにコマンドをセットしなければなりません。そのためには共有ＲＡＭをメインＣＰＵ側に接続しなければなりません。メインＣＰＵ側に接続しているときに、サブＣＰＵが共有ＲＡＭを使用しては困ります。そ

図C－13　共有RAM

図Ｃ－14　ＳＵＢＯＵＴの動作

こで、メインCPUが共有RAMを使用する際は、サブCPUを止めるということをします。サブCPUを止めてしまえば、サブCPUが共有RAMを使用するということもなく、メインCPUが自由に共有RAMを使用することができるわけで、この間に、コマンドなどをセットするわけです。そして、コマンドを設定し終ったら、サブCPUを動かし、共有RAMをサブCPUに返すという手順を取ることになります（図C−14にBIOSのSUBOUTリクエストの場合を例にその動きを図示しました）。

それでは具体的かつ細かい操作について上述の手順にそってみていきましょう。

◀ステップ1▶

まず共有RAMをメインCPU側に接続するためにサブCPUを停止します。

サブCPUは、＄FD05番地のbit 7に1を書き込むことにより停止します。つまり、＄FD05番地に＄80を書き込むと、サブCPUが停止するわけです。プログラムでは

```
LDA  #$80
STA  $FD05
```

とすればサブCPUは停止します。

ところが、どんなときでも勝手にサブCPUを停止してよいというわけにはいきません。サブCPUが前に与えたコマンドをまだ実行中のうちにサブCPUを停止し、共有RAMを書き換えてしまっては、サブCPUが誤動作する原因になります。そのために、サブCPUを停止させようとするときには、事前に、サブCPUがコマンドを実行中であるかどうかを見て、サブCPUがコマンドの実行を終了するまで待ってから、サブCPUを停止しなければなりません。

また、このサブCPUを停止させるという動作は多分にハードウェアに関係している動作のためいろいろなタイミングなどの要素が含まれています。そのため、＄FD05番地に＄80を書き込んでも、すぐにサブCPUが停止するというわけにはいきません。そこで、停止を指示した後には本当に、停止したかどうかを確認する必要があります。

これらのサブCPUの状態は、＄FD05番地を読み出して、そのbit 7をみることによって行うことができます。このbit 7は、サブCPUがコマンド実行中のときと、完全に停止したとき1となり、そうでないときに、0となっています。

これらの手順をサブルーチン形式にして、このサブルーチンを呼び出すことによってサブCPUを停止させるというサブルーチンをプログラムにしたものが図C−15です。

まず最初①にこのサブルーチンで使用して破壊してしまうAレジスタをスタックに退避しています。

②で、サブCPUがコマンド実行中であるかどうかをチェックしています。サブCPUがコマンド実行中（この状態をBUSY：ビジーといいます）のときには、bit 7が1すなわち、＄FD05

〔図C−15 サブCPU停止サブルーチン〕

```
**************
*
* SUBSYSTEM        このルーチンの外で
*    STOP                  HALT  EQU  $FD05
*                  が行われていると仮定
**************
STOP    PSHS   A ··························①：使用するレジスタを退避
STOP1   LDA    HALT ┐
        BMI    STOP1┘·····················②：サブCPUビジーチェック
        ORCC   #$50 ······················③：メインCPU割込み禁止
        LDA    #$80 ┐
        STA    HALT ┘·····················④：サブCPU停止を指示
STOP2   LDA    HALT ┐
        BPL    STOP2┘·····················⑤：サブCPU停止チェック
        PULS   A,PC ······················⑥：終了（サブルーチンから戻る）
```

（＝ＨＡＬＴ：他の場所で定義してあると仮定している）から読み出した値がマイナスになります。ですから、この値がマイナスの間はループし続けて、コマンドを実行し終り次のコマンドを受けつけられる状態（この状態をＲＥＡＤＹ：レディーといいます）、すなわちbit 7が0＝値がプラスになるまで待ちます。

③は、メインＣＰＵの割込みを禁止するということをしています。割込みについてはまだ学習してしないので詳しいことは省略しますが、これをしておかないとサブＣＰＵが誤動作することがあります。今はおまじないと思っていてけっこうです。

それに続く④では、サブＣＰＵの停止を指示しています。これは既に述べたように、＄ＦＤ０５番地のbit 7に1を書き込んでいます。この動作を，ＨＡＬＴ要求といいます。

⑤は、④のＨＡＬＴ要求がサブＣＰＵに受け入れられて正確に停止（ＨＡＬＴ）したかどうかを確認しています（この動作をＨＡＬＴ承認確認といいます）これは、サブＣＰＵが停止すると、＄ＦＤ０５番地のbit 7が1（＝ＢＵＳＹ）になることを利用します。すなわち、bit 7が0の間＝値がプラスの間ループし続けて、bit 7が1になるのを待つことになります。

サブＣＰＵの停止が完了すれば⑥で、①で退避したＡレジスタを復帰すると同時にサブルーチンから戻ります。

◀ステップ２▶

次は、メインＣＰＵに接続された共有ＲＡＭにサブシステムコマンドをセットしなければなりません。転送するデータですが、これは、ＢＩＯＳを使用した使用法のところで出てきた転送データと同じです。このデータを、共有ＲＡＭである＄ＦＣ８０番地からへ転送するわけです。ですからサブシステムコマンドのコマンドコードは＄ＦＣ８２番地にセットされます。

図Ｃ－16がこれを行うプログラムです。Ｘレジスタに転送データの先頭アドレス、Ｂレジスタに転送データの長さ（バイト数）をセットして、このルーチンを呼び出せば、共有ＲＡＭに転送データを転送します。このルーチンは特殊なルーチンではないのですがデータを転送するプログラムの基本ともいえるプログラムですから、少し解説しておきましょう。

①は、このルーチンで使用し破壊するレジスタをスタックへ退避します。

②は、転送先（デスティネーション：destinationといいます）のアドレス＝共有ＲＡＭの先頭アドレスをＵレジスタにセットします。

③は、転送元（ソース：sourceといいます）であるＸレジスタのさすアドレスの内容を、転送先のＵレジスタの指すアドレスへ1バイト転送します。さらに、Ｘ、Ｕレジスタともに命令実行後にオートインクリメントして、次の転送を行うアドレスをさすようにします。

④は、カウンタ（転送するバイト数）をデクリメント（－1）して、次の⑤でカウンタが0でなければループを続けます。

〔図Ｃ－16　共有ＲＡＭへの転送ルーチン〕

```
**************
*                     このルーチンの外で
* TRANSFER TO              SRAM  EQU  $FC80
*   SHARED RAM        が行われていると仮定
*
**************
TRANS   PSHS   A,B,X,U..................①：使用するレジスタを退避
        LDU    #SRAM....................②：Uレジスタに共有RAMの先頭アドレスをセット
TRANS1  LDA    ,X+ ┐
        STA    ,U+ ┘....................③：1バイトを転送
        DECB....................................④：カウンタをデクリメント
        BNE    TRANS1...................⑤：カウンタが0になるまで転送
        PULS   A,B,X,U,PC...............⑥：終了
```

⑥は①でスタックに退避したレジスタを復帰すると同時にサブルーチンから戻ります。

◀ステップ3▶

さて、以上で共有RAMへのサブシステムコマンドの書込みが終了したので、共有RAMをサブCPUに返して、サブCPUを動作させてコマンドを実行させます。

サブCPUの停止を解除して動作させるには停止するときに1にした$FD05番地のbit 7を0にします。つまり、$FD05番地に0を書き込めばよいわけで、プログラムでは

CLR $FD05

とすればよいわけです。これをサブルーチンにしたのが図C—17のプログラムです。

①で、サブCPUの停止を解除します。次の②は、サブCPU停止ルーチン(図C—15)の③のメインCPU割り込み禁止に対応する処理で、メインCPUの割り込みを許可します。これも、今はおまじないだと思っていてけっこうです。そして③でサブルーチンから戻るというわけです。

以上の3ステップがBIOSを使用せずにサブCPUにサブシステムコマンドを実行させる手順です。BIOSのSUBOUTリクエストもこれとほぼ同じことを行っています。それでは、図C—7と同じことを、BIOSを用いずに行うようにしたのが図C—18です。使用したサブルーチンは、各ステップで解説したものと同じですので、メインルーチン(6～11行め)についてのみ解説しましょう。

まずXレジスタにサブシステムコマンドが設定されているアドレスをセットします。ここでちょっとした技法を用いています。これは、サブシステムコマンドの長さを、サブシステムコマンドの書かれているアドレスの最初に書いていき(53行め)、

LDB ,X+ (方法1)

でBレジスタにセットするようにしています。ここは、

LDX #SCOM (方法2)
LDB #ECOM−SCOM

というようにしてもいっこうに構わないのですが、表示する文字数を変えたいという場合など、方法1では、モニタなどで、$5035番地付近のみを集中的に変更するだけでよいのですが、方法2を取ると、変更する所が散在(プログラムの先頭付近と後尾付近に分散している)してしまうため変更が繁雑になります。

それ以降については解説の必要はないでしょう。

ちなみに、このプログラムは6行めを

START LEAX SUBCOM, PCR

とすると、ポジションインディペンデントになります。

以上でBIOSのSUBOUTに対応する部分については解説を終りました。ではSUBINに対応した動作はどうしたらよいのでしょうか。

この説明は、だいぶSUBOUTの解説と重複してしまうので、ここでは実際のプログラムを通して、それを学習します。

図C—19は、前章7節の図B—25のプログラムをBIOSを使用しないで、実行させようというものです。用いるサブシステムコマンドはINKEYコマンドで、そのパラメータおよび復帰情報

〔図C—17 サブCPU停止解除ルーチン〕

```
**************
*
* SUBSYSTEM
*     RUN
*
**************
RUN     CLR     HALT ·························· ①:サブCPU停止解除
        ANDCC   #$AF ·························· ②:メインCPU割込み 許可
        RTS ··································· ③:終了
```

〔図C－18 SYMBOLコマンド〕

```
  1   5000                          ORG     $5000
  2
  3                  FD05   HALT    EQU     $FD05
  4                  FC80   SRAM    EQU     $FC80
  5
  6   5000 8E    5035       START   LDX     #SUBCOM
  7   5003 E6    80                 LDB     ,X+
  8   5005 8D    05                 BSR     STOP
  9   5007 8D    1E                 BSR     TRANS ……………………ステップ1
 10   5009 8D    16                 BSR     RUN ………………………ステップ2
 11   500B 39                       RTS ……………………………………ステップ3
 12
 13                             **************
 14                             *
 15                             * SUBSYSTEM
 16                             *    STOP
 17                             *
 18                             **************
 19   500C 34    02             STOP    PSHS    A
 20   500E B6    FD05           STOP1   LDA     HALT
 21   5011 2B    FB                     BMI     STOP1
 22   5013 1A    50                     ORCC    #$50
 23   5015 86    80                     LDA     #$80
 24   5017 B7    FD05                   STA     HALT
 25   501A B6    FD05           STOP2   LDA     HALT
 26   501D 2A    FB                     BPL     STOP2
 27   501F 35    82                     PULS    A,PC
 28
 29                             **************
 30                             *
 31                             * SUBSYSTEM
 32                             *    RUN
 33                             *
 34                             **************
 35   5021 7F    FD05           RUN     CLR     HALT
 36   5024 1C    AF                     ANDCC   #$AF
 37   5026 39                           RTS
 38
 39                             **************
 40                             *
 41                             * TRANSFER TO
 42                             *    SHARED RAM
 43                             *
 44                             **************
 45   5027 34    56             TRANS   PSHS    A,B,X,U
 46   5029 CE    FC80                   LDU     #SRAM
 47   502C A6    80             TRANS1  LDA     ,X+
 48   502E A7    C0                     STA     ,U+
 49   5030 5A                           DECB
 50   5031 26    F9                     BNE     TRANS1
 51   5033 35    D6                     PULS    A,B,X,U,PC
 52
 53   5035 14 ══ ↙20            SUBCOM  FCB     ECOM-SCOM
 54   5036 00 00 19 05          SCOM    FCB     0,0,$19,5
 55   503A 00 00 05 03                  FCB     0,0,5,3
 56   503E 00A0 0000                    FDB     160,0
 57   5042 07                           FCB     7
 58   5043 5A 49 47 20                  FCC     /ZIG ZAP/
      5047 5A 41 50
 59                  504A   ECOM    EQU     *
 60
 61                                 END     START

 0 ERROR(S) DETECTED

   SYMBOL TABLE:

 ECOM   504A   HALT   FD05   RUN    5021   SCOM   5036   SRAM   FC80
 START  5000   STOP   500C   STOP1  500E   STOP2  501A   SUBCOM 5035
 TRANS  5027   TRANS1 502C
```

〔図C－19　INKEYコマンド〕

```
 1   5000                            ORG     $5000
 2
 3                FD05  HALT         EQU     $FD05
 4                FC80  SRAM         EQU     $FC80
 5
 6   5000 8E    5048    START        LDX     #SUBCOM  ┐……サブシステムコマンドの先頭
 7   5003 E6    80                   LDB     ,X+      ┘  アドレスとバイト数
 8   5005 8D    18                   BSR     STOP
 9   5007 8D    31      LOOP         BSR     TRANS……………ステップ1
10   5009 8D    29                   BSR     RUN……………ステップ2
11   500B 8D    12                   BSR     STOP……………ステップ3
12   500D B6    FC83                 LDA     SRAM+3
13   5010 81    20                   CMPA    #$20       SPACE ┘……チェック
14   5012 26    F3                   BNE     LOOP
15   5014 B6    FC80                 LDA     SRAM        ┐
16   5017 8A    80                   ORA     #%10000000  ┘……Ready Request
17   5019 B7    FC80                 STA     SRAM
18   501C 8D    16                   BSR     RUN
19   501E 39                         RTS
20
21                          **************
22                          *
23                          * SUBSYSTEM
24                          *    STOP
25                          *
26                          **************            ┐
27   501F 34    02          STOP     PSHS    A        │
28   5021 B6    FD05        STOP1    LDA     HALT     │
29   5024 2B    FB                   BMI     STOP1    │
30   5026 1A    50                   ORCC    #$50     │ 既
31   5028 86    80                   LDA     #$80     │
32   502A B7    FD05                 STA     HALT     │ に
33   502D B6    FD05        STOP2    LDA     HALT     │
34   5030 2A    FB                   BPL     STOP2    │ 解
35   5032 35    82                   PULS    A,PC     │
36                                                    │ 説
37                          **************            │
38                          *                         │ し
39                          * SUBSYSTEM               │
40                          *    RUN                  │ た
41                          *                         │
42                          **************            │ サ
43   5034 7F    FD05        RUN      CLR     HALT     │
44   5037 1C    AF                   ANDCC   #$AF     │ ブ
45   5039 39                         RTS              │
46                                                    │ ル
47                          **************            │
48                          *                         │ ｜
49                          * TRANSFER TO             │
50                          *    SHARED RAM           │ チ
51                          *                         │
52                          **************            │ ン
53   503A 34    56          TRANS    PSHS    A,B,X,U  │
54   503C CE    FC80                 LDU     #SRAM    │ 群
55   503F A6    80          TRANS1   LDA     ,X+      │
56   5041 A7    C0                   STA     ,U+      │
57   5043 5A                         DECB             │
58   5044 26    F9                   BNE     TRANS1   │
59   5046 35    D6                   PULS    A,B,X,U,PC ┘
60
61   5048 04                SUBCOM   FCB     ECOM-SCOM
62   5049 00 00 29 03       SCOM     FCB     0,0,$29,%00000011
63                504D      ECOM     EQU     *
64                                                    ……ウェイトフラグ＝1 ┐ 詳しくは
65                                   END     START    ……リセットフラグ＝1 ┘ 図C－19
                                                                             を参照
 0 ERROR(S) DETECTED
                                                      ……INKEYコマンド
```

```
SYMBOL TABLE:

ECOM    504D    HALT    FD05    LOOP    5007    RUN     5034    SCOM    5049
SRAM    FC80    START   5000    STOP    501F    STOP1   5021    STOP2   502D
SUBCOM  5048    TRANS   503A    TRANS1  503F
```

は図C－20に示すとおりです。

このプログラムでも21行め～59行めまでのサブルーチンは、既に解説したものと全く同じです。

まず6～10行めまではSUBOUTのときと全く同じに、サブCPUにコマンドを与えて実行させます。サブCPUはメインCPUが10行めを実行すると同時に（正確には43行め）コマンドを実行し始めます。

次にメインCPUはサブCPUから実行結果を受け取るべく、サブCPUを停止にかかります。プログラムをみると、サブCPUを動かしてすぐに停止させているから、サブCPUはコマンドを実行する間もなく止められてしまうのではないかと思われるかもしれません。しかし、STOPのサブルーチンでは、28～29行めで、サブCPUがコマンドの実行が終るまで待ちますので、問題は生じません。

サブCPUの実行結果は、共有RAMにセットされているので、その結果をメインCPUで読み取ります。ここでは、相対値3＝$FC83に押されたキーのアスキーコードが入っているのでこれを読み出し、スペースがどうかをチェックしています。そしてスペースでなければ、再度、サブシステムコマンドを実行させるためにLOOPへ行きます。

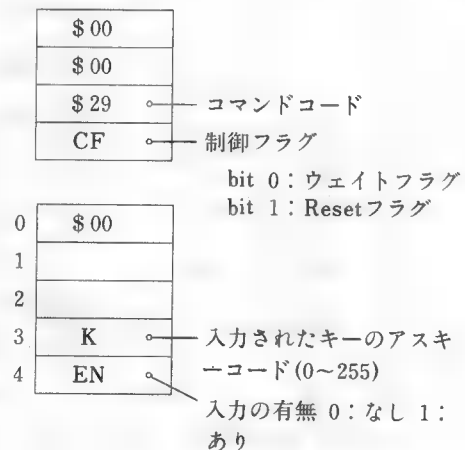
さて、ここまでは別に難しい点もないと思います。問題となるのは、この次です。つまり、スペースが押されていた場合です。この時には、サブCPUの停止を解除して、サブCPUに次のコマンドまで待機してもらうようにします。サブCPUの停止を解除するだけでしたら、

BSR RUN

だけでよいはずですが、ここではそううまくいきません。

サブCPUはハード的に自分がメインCPUに停止させせられたことを認識するようになっています。しかしサブCPUはソフト的にそのことを知ることはできないようになっています。つまり、サブCPUは自分が停止させられたことを体ではわかっても頭で認識することができないわけです。

通常サブCPUが停止させられるのは、コマンドを与えられるときです。この場合には、ハード的にはもちろんソフト的にも停止させられてコマンドがセットされたこともわかります（共有RAMに停止前と違う値がセットされていることをみれば理解できるでしょう）。

しかし、サブCPUが停止させられたにもかかわらずコマンドを与えられなかったときは問題です。例えば、コマンドの実行結果をみるためだけにサブCPUを止めて、新しいコマンドをセットしない場合などです。この場合、サブCPUはハード的には停止させられたことを認識しているにもかかわらず、ソフト的には共有RAMにコマンドがセットされているわけでもないので、停止させられたことを知ることはできません。こうなる

図C－20 INKEYコマンド

と、次にメインCPUがコマンドを与えるべくサブCPUを停止させようとしても、サブCPUのハードとソフトの認識の違いからBUSYのままになっているので、停止させることができなくなってしまいます(この状態をデッドロック：deadly embraceといいます)。

そこで、新しくコマンドをセットしないのに(実行結果を得るためなどの目的で)サブCPUを停止させたときは、停止を解除する前に、共有RAM上のレディ要求フラグ（Ready Request）をセットしておくことによって、サブCPUがソフト的に、停止させられたけれどもコマンドはセットされなかったということを認識させるようにします。

この処理をしているのがプログラム中の、15行め〜17行めです。レディ要求フラグは、相対値０＝＄FC８０番地のbit７となっていて、新たにコマンドをセットしないときには、このbit７を１にしておくわけです。この際＄FC８０番地のbit０〜bit６は変更してはいけません。

以上で、BIOSを用いずに、直接サブCPUをコントロールする方法を学習しました。ここではとりあえず必要最低限と思われることだけを解説しました。サブCPUをもっと活用するには、より詳しくサブCPUの働きを知らなければなりません。より深く知りたい方は、解析マニュアルのフェーズIIIサブシステム編を参照するとよいでしょう。

第 $ D 章

プログラミング実践

——ゲームプログラムを題材に——

1. 課題の選択

これまで、私たちは、基礎編でマシン語命令の使い方、実践編の前半でFM—7シリーズのマシン語プログラミングにおける必須事項であるBIOSとサブシステムの使い方を学習してきました。そこで、この章ではこれまで学習したマシン語の知識を土台にして、ある程度長いプログラムを作成していくことにします。これにより、より実践的なマシン語プログラミングの方法が理解できるようになります。

実践的なプログラミングとなると、単なる命令の実行例というようなプログラム（基礎編のサンプルプログラムなど）では、意味がありません。またFM—7シリーズの特徴を活かしたプログラムでなければ、単なる6809CPUの理解にとどまってしまい、本書の主旨に反します。

そこで、課題としては、サブシステムを活用しかつ速度が要求されるという点を考慮して、リアルタイムゲームを取りあげます。よくマシン語の解説書などでは、ビジネスプログラムに似たものなどのサンプルを用いることが多いのですが、FM—7シリーズの特徴を活かすとなるとやはりゲームだと考えられます。このプログラミングで培かわれた技術は、ビジネスプログラムやユーティリティプログラムの作成などに十分活かすことができます。

ゲームを作るとなると、ゲームのアイデアとゲームに出てくるキャラクタとが、そのおもしろさに大きく影響するので、その点の構想を練るのに十分時間をかけなければなりません。しかし、ここではあくまで、マシン語学習のためのプログラムということに割り切って、単純化したゲームを取りあげます。おもしろいものに改造することは、読者の方々への今後の課題ということにしておきます。

2. ゲームの仕様

それでは、このゲームの仕様を述べていくこと

にしましょう。

まずゲームの形式は、リアルタイムゲームの原形ともいえる宇宙もののゲームとします。この形式のゲームはいろいろなジャンルのゲームの基本となる要素を多く持っており、このゲームを理解しておけば、他のジャンルのゲームも比較的容易に理解ができると思います。簡単にまとめると、上から降りてくる敵をキャノン（砲台）から発射する破壊光線によって、破壊していくというものです。ただし、このゲームでは構造を複雑なものとしないようにするために敵は発砲しないということにします。

敵の数は、常に4個ですが、必要に応じてプログラムの一部を書き換えることによって、その数を増やせるようにしておきます。そして敵の動きは完全にランダムにしておき、もし、敵がせまってきたのにキャノンが避けることができずにあたってしまった時点で、ゲームオーバーにすることにします。

味方であるキャノンの数は、1台だけにします。というのは、ゲームセンターの場合と違い、パソコンを用いたゲームであれば、好きな回数だけできるので、何台もキャノンが必要ではないからです。このキャノンの数は1台ですが、キャノンから発射する破壊光線は、10発まで連発できるようにします（この点はプログラミングの際には一つの大きな難関となるかもしれませんが)。

画面に4つの敵と味方のキャノンだけでは若干さびしいので、画面背景には、いくつかの星が上から下へ流れるようにしましょう。また、ゲーム中になにも音が出ないのは、ゲームらしくないので、ここでは、キャノンから破壊光線を発射したときと、みごと破壊光線が命中し、敵が破壊されたときに、ちょっとした効果音を入れるようにします。

図D－1　ゲームの様子

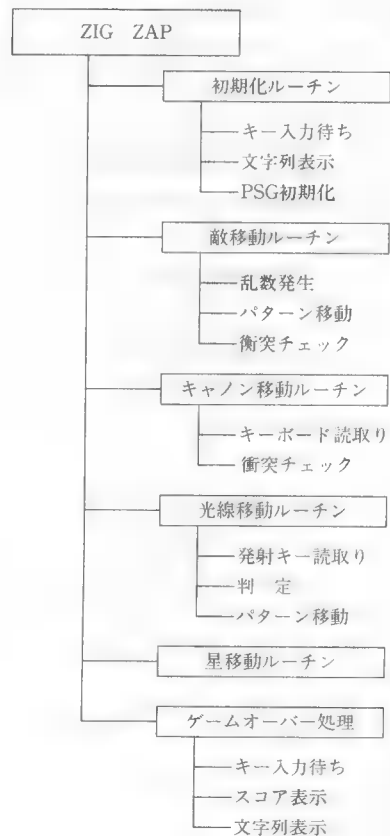
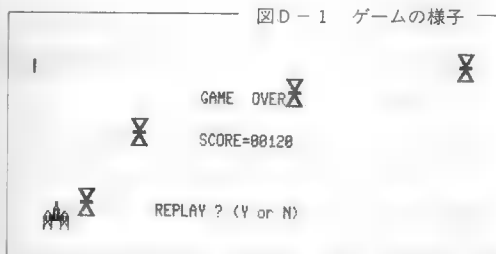
さて最後にこのゲームの名前ですが、ジグザグに降りてくる敵をやっつけようという意味を合成して

ZIG ZAP

という名前をつけました。

3. ゲームの構造

プログラムの仕様が決定すれば、次の段階は、そのプログラムの構造の決定です。

プログラム構造といっても大げさな物を想像することはありません。仕様にそったプログラムを作成するには、実際にしなければならないことを、プログラムの仕様を眺めつつイメージしていけば

図D－2　構造図

容易に作成することができます。

まず思いつくのは、砲台（キャノン）の移動を行うルーチンです。このルーチンでは、キー入力とパターンの表示をしなければいけません。

次に、敵の移動を行うルーチンも必要です。このルーチンではパターン表示の他に、移動を決定するために乱数を用いるので乱数を生成するルーチン、それに、敵がキャノンに衝突していないかをチェックするルーチンも必要です。

また破壊光線のためのルーチンも必要です。これは発射キーの読み取り、当ったかどうかの判断、光線のパターンの移動などを行わなければいけません。特にこのルーチンに関しては、破壊光線の連射ということを考えなければいけないので、他のルーチンに比べると少々難かしいかもしれません。

画面の背景には星が流れることになっていますから、その星の移動を行うルーチンも必要です。さらに、ゲーム開始時の初期化ルーチンやゲームオーバー時のゲームオーバー処理を行うルーチンなども必要です。

このように、プログラムに必要な事項を仕様書をみながら考えていくと、このプログラム構造は比較的簡単に考えることができるでしょう。だいたいの感じがつかめたら、図D—2にあげた構造図を書いてみると整理されます。ラフにでも紙にこのような構造図を作っておくと後々の作業に意外と便利です。

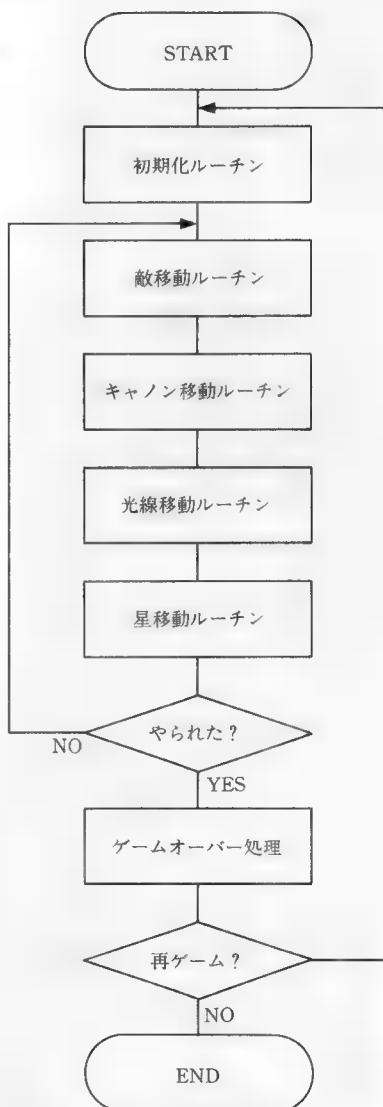
ここまでくれば、図D—3に示すゼネラルフローチャートも同時に作成してみましょう。このゼネラルフローチャートは直接のプログラミング（特にアセンブリ言語への書きくだす場合など）において、役立つというものではありませんが、全体の構成を正確に把握するという意味では有意義なものです。

図D−3　ゼネラルフローチャート

4. プログラミングの方法

さて、ここまででプログラムの中でどんな動作をしなければならないか、またどういう順序で行わなければならないかということが明らかになりました。

ここから先のプログラミングには、二種類の方法があります。

まず第一の方法は、**ボトムアッププログラミング**と呼ばれる方法です。この方法ではまず構造図にある必要なルーチンを、1つのルーチンとしての意味を失わないというところまで機能を分割していきます。すると、1つのルーチンとして一つの機能を持ったサブルーチンがいくつもできてくることになります。例えば、このゲームでいうな

らば、キー入力待ちやＰＳＧ初期化などのルーチンです。そこで、次により詳しいマシン語対応フローチャート（またはそれに近いもの）を作成する際、まず、この様な、下位の（構造図でいけば、一番末端の）サブルーチンから作成していき、次第に上位のサブルーチンを作成していきます。そして、最後にそれらをまとめた形でメインルーチンを作成し、プログラムを完成させるという方法です。つまり、いろいろな種類の部品を一つずつ作成していき、それを積み重ねて、プログラムを完成させるというわけです。

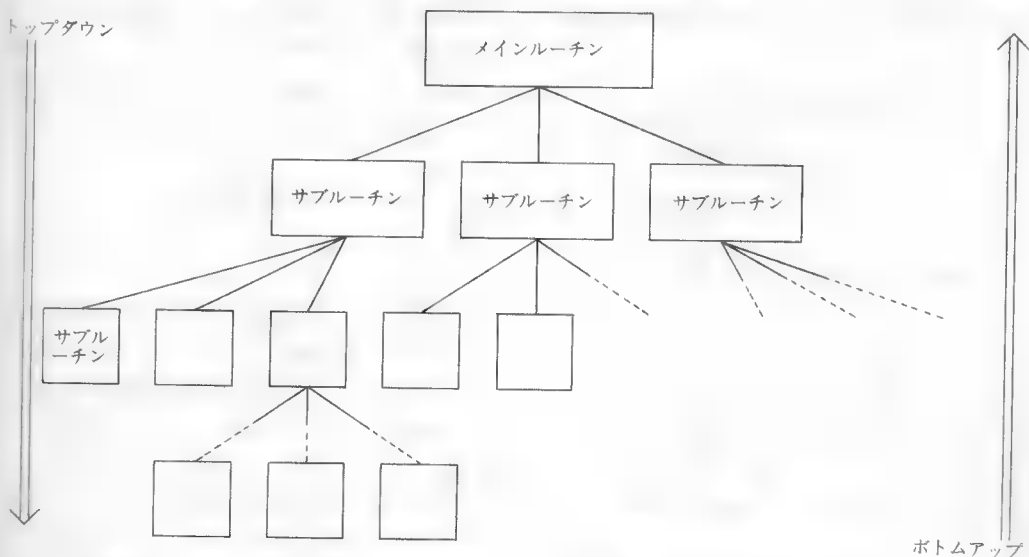
このボトムアッププログラミングというのは関数を一つ一つ定義していくことによってプログラミングを行う、ＰＡＳＣＡＬやＦＯＲＴＨ（いずれも高級言語の一種）でよく用いられるプログラミングの方法です。

この方法の利点は、１つ１つのサブルーチンができあがるたびに、それを順々にテストしながらプログラミングすることができるという点です。例えば、キー入力待ちのルーチンを作ったとすれば、それをテストするごく簡単なテストルーチンを作ってやり、そのルーチンをテストして間違いがないかどうかを確認して、次のサブルーチンの作成に移るというわけです。順番に、作ったものからテストしておけば、完成したときのデバックもだいぶ楽になります。

一方、この方法では、各サブルーチンに構造を分割する際、そのプログラムでのデータ構造をきちんと決定しておかないと、各ルーチンの間で誤解が生じてしまうという点に注意しなければなりません。また、部品を積み重ねていって完成させるというプロセスは、まず大まかに内容を把握してから、細いところを考えていくという人間の物事のとらえ方と逆になっているという点で、多少の慣れを必要とするかもしれません。

第２の方法は、**トップダウンプログラミング**と呼ばれる方法です。これは、ボトムアッププログラミングとは、全く逆に、まず大まかな処理を記述し、順々に細かな処理を記述していく方法です。つまり、メインルーチンを最初に作ってしまい、その中で必要なサブルーチンを順々に上位から下位へと作成していくわけです。

この方法では、人間の物事のとらえ方と一致しているので、考え方はわかりやすいと思います。

このように、プログラミングには、ボトムアッププログラミングとトップダウンプログラミングという２つの方法があります。このどちらの方法でプログラミングをするかは、使用する言語（例えば先に例にあげたＦＯＲＴＨなどでは、ボトムアッププログラミングしかできない構造になって

図D－4 ボトムアップとトップダウン

います）と、各個人の好みによります。いちがいにはいえませんが、ＢＡＳＩＣでプログラムを組むことに慣れている人の場合、トップダウンプログラミングの方を取る人が多いようです。そこで、ここでは、トップダウンプログラミングを中心にしてゲームを作成していくことにします。

5. データ構造の決定

プログラムの構造が決定したら、次にデータの構造を決定します。

このゲームでデータとなるのは、キャノンの位置、敵の位置、そして破壊光線の位置です。

まずキャノンの位置ですが、このキャノンは左右にしか移動しないので、画面上での位置はX座標だけを記憶しておけばよいわけです。ここで、一つプログラムを複雑化しないための制約を作ります。キャノンの位置や敵、破壊光線の位置を記憶する場合、その位置を、X座標０～639、Y座標０～199で記憶すると、位置の記憶にX座標として２バイト用いることになり、処理が複雑とならざるをえません。また、移動の処理の際、グラフィック座標で１ドットずつ動かしていたのでは実行に時間がかかりすぎ、ゲームらしくなくなってしまいます。そこで、キャノンや敵、破壊光線は、80桁25行画面でのキャラクタ単位で移動することにします。こうすると、X座標の記憶も１バイトですみますし、移動処理も手速く行うことができます。これにより、移動では多少ぎこちなさが伴うことになりますが、そこは目をつぶることにします。結局、キャノンの位置の記憶には１バイトが必要なわけです。

次に破壊光線の位置の記憶にはどうしたらよいでしょうか。破壊光線の位置は、X座標とY座標で表せます。さて、この破壊光線は最大10発まで連射できるようにするので、10組分のX、Y座標の記憶が必要です。さらに考えなくてはならないのは、常に画面上に10個の破壊光線があるわけではないということです。ですから、10個の破壊光線それぞれに、その破壊光線が有効なのか（画面上にあるか）を記憶しておく場所（有効フラグとでもしましょう）をもうける必要があります。これらの事から、図Ｄ―5にあるようなデータ構造とします。結局、３×10＝30バイトの領域を使用するわけです。ここで有効フラグは○（マル）と×（バツ）で示してありますが、実際のプログラムではそれぞれに値を割りあてます。値の割りあては、光線移動ルーチンで行うことにします。

さて最後に、敵の位置のデータ構造です。これには、いろいろな考え方があり、採用する考え方によって、全くデータ構造がかわります。そこで、この敵の位置のデータ構造については、作成したプログラムのデータ構造について敵移動ルーチンの所で詳しく解説することにして、ここでは省略します。ここでは練習と思って各自で自分なりのデータ構造を考えてみると良いでしょう。

（10個）

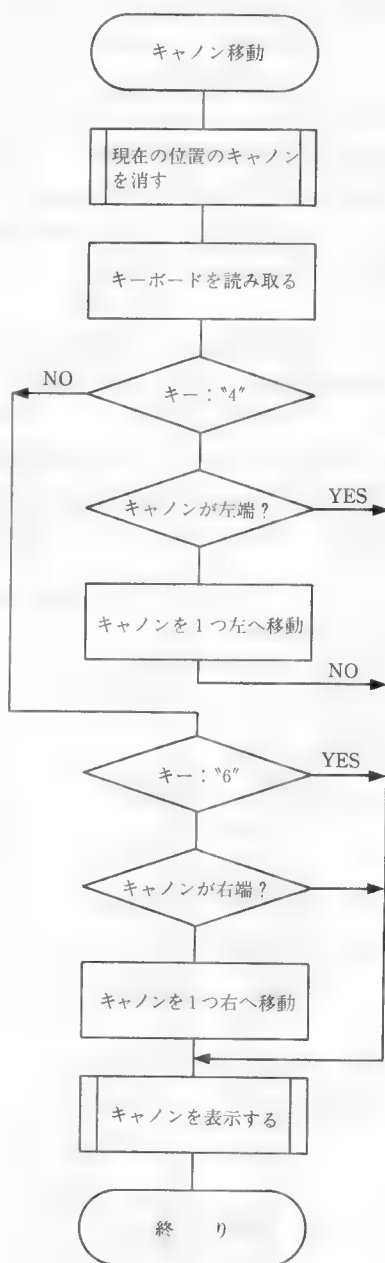
有効フラグ	X座標	Y座標
○	10	16
×	?	?
○	78	5
○	60	15
×	?	?
×	?	?
○	32	12
×	?	?
×	?	?
×	?	?

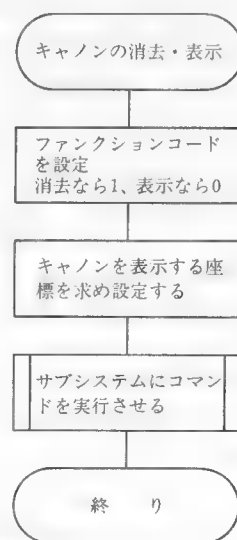
図Ｄ－５　破壊光線のデータ構造

6. キャノン移動ルーチン

このルーチンの仕事は、キーボードの入力によってキャノンを移動することです。もう少し詳しくいえば、４のキーが押されていれば左に、６のキーが押されていれば右に、キャノンを移動させればよいわけです。

このルーチンのフローチャート（ディーテル（くわしい）フローチャート）は、図Ｄ―6のようになります。実はこのフローチャートは、後にあげる実際のマシ

ン語プログラムにかなり近い形でかかれています。ですから、読者の中には、こんなフローチャートはすぐには書けないという人が大半だと思います。実際になにもないところからフローチャートを作る場合には、もっと入り組んだり、横に大きく広がった形になってしまうものです。まずは、そういう形であっても実際に書いてみることが重要です。個人的に使用するプログラムであれば（雑誌に投稿したり、他人にみせたりするのでなければ）このフローチャートは自分にわかるように書けばよいのですから、その記述形式をＪＩＳ（日本工業規格）のフローチャートの書き方にあわせる必要はありません。もちろんフローチャートにかわる独自の記述形式があればそれを用いてもかまいません。だからといってないがしろにしてはいけません。この段階の文書（ドキュメント）は、デバッグの際や、後々のプログラムの改良などの際には重要な資料となるので、**必ず後で見てわかるように書かなければいけません。**

図D－6　キャノン移動ルーチン

このフローチャートを書くときに注意しなければならないことをいくつかあげておきましょう。まず、１つのフローチャートにそのルーチンのする仕事を全て押しこめるということを考えないことです。この例の場合、キャノンの消去や表示は、別のルーチン（下位ルーチン）にまかせるつもりで、その処理は詳しく記述しません。もし、このようにしないと、一つのフローチャートが長くなってしまい、見通しが悪くなると同時に、頭の混乱を招くことになります。特に基準といったものがあるわけではありませんが、フローチャートを書いていて、１つのルーチンがＢ５（本書の大き

図D-7　キャノンの消去・表示

さ）の紙に収まりきれないようであれば、そのルーチンの一部をサブルーチンとして、詳しく記述しないようにします。このように、１つのフローチャートが２枚の紙にまたがることのないようにします。フローチャートが２枚にまたがると、そのフローチャートを把握することは１枚に収まったたときに比べて、格段に難しくなります。

このこととも関係しますが、もしすぐに図D—6のレベル（詳しさのレベル）のフローチャートを書けなければ、粗いフローチャートをまず書いてみて、それから順々に段階を追って細かく書いてみると､比較的容易に図D—6のレベルまで達することができると思います。

同様にしてキャノンの消去、表示のルーチンのフローチャートも作成します。(図D—7)

さて、ディーテルフローチャートが完成すればもう実際の**コーディング**（アセンブリ言語で書きくだすこと）ができるようになっています。もし不安であれば、その前に**マシン語対応フローチャート**のようなものを書いてみるのもよいでしょう。

図D—6、7のフローチャートにそってコーディングしてプログラムしたのが図D—8のプログラムです。まずは、そのコーディングの際の注意点をいくつか述べておきましょう。

ここまでで作成したフローチャート類は、それをみれば、ある程度感覚的にその処理内容をわかるようになっていました（実際、フローチャートというものはそのために開発されたものです)。しかし、コーディングして得られるアセンブリ言語のソースプログラムは、それを見て処理内容を感覚的に理解するのはまず不可能です。このように人間にはこのコーディングという作業は、わかりにくさを増す作業ともいえます。ところで、アセンブラの使い方の項で述べましたが、アセンブリ言語で書かれたソースリストには、ＢＡＳＩＣなどと同じく、注釈を付加することができます。そこで、わかりにくさをおさえるためにも、この注釈を大いに活用すべきです（本書では本文中や活字による解説を各プログラムに付けている関係上、ソースプログラム上にあまり注釈を用いていません。これは、プログラムリストを不必要に複雑にしないための処置です。ですから、読者の方が、本書に掲載されているマシン語プログラムをソースプログラムの形で入力する際や、独自のプログラムを作成する際は、英字やカナ文字などで、十分に注釈文をつけることが望まれます)。

また、これはすでにアセンブラの使い方の項で述べたことですが、プログラム中で用いる定数やＩ／Ｏのアドレスなどの値は、その意味を十分に推しはかることのできるような名前（シンボル）を付けて、 EQU擬似命令で定義しておくことが、良いコーディングのひけつです。というのは、名前で定数他を参照するようにすると、もしその定数を変更しなければならなくなった場合にも１箇所の変更ですみますし、なによりもプログラムがわかりやすくなります。

さて、話がそれてしまいましたが、元に戻って、図D—8のプログラムの内容を解説していきましょう。

①は、現在の位置のキャノンを消す処理を行っています。キャノンの消去と表示は、同一のサブルーチンで行うことにして、ここでは消去である

〔図D－8　CANNON〕

```
*
* CANNON MOVE ROUTINE
*
CANNON    EQU    *
       ┌ LDA    #1
     ① └ BSR    CANPUT
CAN1   ② LDA    KEYBRD
       ┌ CMPA   #'4
     ③ └ BNE    CAN2
       ┌ TST    CANX,PCR
     ④ └ BEQ    CAN3
       ⑤ DEC    CANX,PCR
       ⑥ BRA    CAN3
CAN2   ┌ CMPA   #'6
     ⑦ └ BNE    CAN3
       ┌ LDB    CANX,PCR
     ⑧ │ CMPB   #77
       └ BHS    CAN3
       ⑨ INC    CANX,PCR
CAN3   ⑩ CLRA
CANPUT ⑪ LEAX   CANPTN,PCR
       ⑫ STA    13,X
       ┌ LDA    CANX,PCR
     ⑬ │ LDB    #8
       └ MUL
       ┌ STD    4,X
     ⑭ │ ADDD   #3*8-1
       └ STD    8,X
       ⑮ LBRA   TRANS
```

ことを、Aレジスタに1をセットすることによってパラメータとしてサブルーチンに伝えています。

②は、キーボードを読み取っています。FM—7シリーズでは、このKEYBRD（$FD01。他の所で定義する）番地に最後に押されたキーのアスキーコードが保持されています。②のようにするとAレジスタに最後に押されたキーのアスキーコードが入ります。ここで、最後に押されたキーという表現をしたのは、FM—7シリーズでは、キーが押されていない（押された後に離された）ことを知ることはできない構造になっています。このあたりのことは、図D—9のBASICプログラムを走らせてみればわかると思います。

③では、押されていたキーが“4”でなければ、CAN2へ分岐します。

④では、現在のキャノンの位置がX＝0でないかどうかを調べています。X＝0のときには、左へ移動することはできないのでCAN3へ分岐して移動処理をパスします。

⑤は、左に移動させるルーチンです。キャノンのX座標を1減らしています。⑥で左移動の処理は終りです。

⑦は、“4”でなかった場合に、押されたキーが“6”であるかをチェックします。“6”でなければCAN3へ分岐します。

⑧は、右へさらに移動できるかをチェックするルーチンです。キャノンの大きさは、横80字のキャラクタにして横3文字分ですから、X≧80－3＝77であるかをチェックしています。

⑨は、右に移動させています。

⑩からが、このルーチンの終了処理にあたります。⑩では、キャノンの消去・表示ルーチンへ、Aレジスタを0（表示）にして移ります。ここは通常ならば

```
CLRA
BSR CANPUT
RTS
```

から、RTS命令をBSR命令とあわせて、

```
CLRA
BRA CANPUT
```

とすべきですが、次にすぐCANPUTのルーチンを置くことにより、BRA命令を省略しています。

そして⑪からが、キャノン消去・表示のサブルーチンとなるわけです。

⑪では、Xレジスタにサブシステムコマンドとしてサブ CPUに渡そうとしているパラメータのある先頭アドレスをセットしています。そのパラメータが図D—10です。構造としては図D—11の構成を取っています。ここでは⑫で、ファンクシ

〔図D—10　CANPTN〕

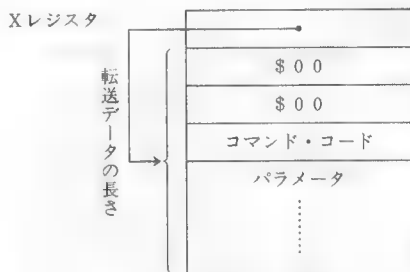
```
*
* CANNON PATTERN
*
CANPTN ①FCB     CANP2-CANP1
CANP1   ┌FCB     0,0,$1C
       ② FDB     0,23*8,0,24*8+7
        └FCB     5,0,8*6
        ┌FCB     %00000000,%000110
       ③ FCB     %00000000,%00011
        ↓ FCB     %00000000,%00011
          FCB     %00000000,%00100
          FCB     %00000000,%01111
          FCB     %00011000,%01100
          FCB     %00100100,%011001
          FCB     %01000010,%011001
          FCB     %10100101,%111001
          FCB     %10100101,%1110011
          FCB     %10100101,%1111111
          FCB     %10011001,%000000
          FCB     %10100101,%000000
          FCB     %11000011,%00000
          FCB     %10000001,%0000
          FCB     %10000001,%0000
CANP2     EQU     *
```

〔図D—9　$FD01のテストプログラム〕

```
100 ' test of I/O address $FD01 (KEYBRD)
110 PRINT
120 C=PEEK(&HFD01)
130 IF C<&H20 OR C=&H7F THEN C=ASC(".")
140 PRINT CHR$(C);
150 GOTO 120
```

ョンコード（0なら表示＝ＰＳＥＴ、1ならば消去＝ＰＲＥＳＥＴ）をセットします。そして、⑬で、キャノンのＸ座標（0～79）を8倍することによりグラフィックでのＸ座標（0～639）を算出しています。これは、ＰＧＢＬＫ1コマンドの開始位置のＸ座標として設定され、さらに、これに、3×8－1＝23を加えた値を終了位置のＸ座標としてセットしています。つまり、キャノンのパターンの左上と右下の座標をセットしたわけです。

⑮で、このパラメータをサブＣＰＵに転送するルーチンを呼び出して、このルーチンの処理を終了します。

図Ｄ―10について解説しておくと、①がサブＣＰＵに転送するパラメータ（コマンドコードなども含む）の長さ、②がコマンドのパラメータ、③がその内のキャノンのパターンデータを示しています。キャノンのパターンは、横80字のキャラクタに換算して3×2、ドット数にして24×16ドットの大きさです。

図D－11　TRANSにおけるパラメータ

7. 星移動ルーチン

簡単なルーチンから巧略していくことにします。ここでは、星移動ルーチンを取りあげます。このルーチンのディーテルフローチャートが図Ｄ―12です。言葉で簡単に書けば、一つ一つの星についてＹ座標を増加させ、画面をオーバーしたら上に戻すという作業を全ての星について行うということになります。

それではプログラム（図Ｄ―13）をみていきましょう。

①では、まず星を消去しています。ここでは、星の消去・表示をサブルーチンとしているので、消去を示す1（ファンクションコードのＰＲＥＳＥＴ）をＡレジスタにいれて、サブルーチンを呼び出します。

②では、最初の星を選択しています。ここで星の数は、サブシステムコマンドのＰＯＩＮＴで指定できる最大の数の20コとしています。Ｙレジスタにその20を代入し（Ｙレジスタをカウンタとして用いる）、Ｘレジスタには、最初の星のＹ座標が格納されているアドレスであるＳＴＲＰＴＮ＋7をセットしています。

③では、星のＹ座標をＤレジスタにとり出してそれに4を加えることにより、星を下に4ドットだけ移動させています。

④では、移動後のＹ座標が200をこえていないかどうかをチェックし、200以上のとき（画面からはみだしたとき）には⑤で、200を引くことにより、画面の上へ戻しています。そして、⑥で変更されたＹ座標を元のところへ設定し直しています。

ここで、少しＰＯＩＮＴコマンドについて述べておきましょう。このＰＯＩＮＴコマンドのパラメータは図Ｄ―14になっています。ここでは、表示点数に20を指定しています。このパラメータはプログラム中の⑰で、ＦＣＢ命令によって設定されています。このうち、Ｙ座標とファンクションコードは仮の値で、これらの値は、プログラム実

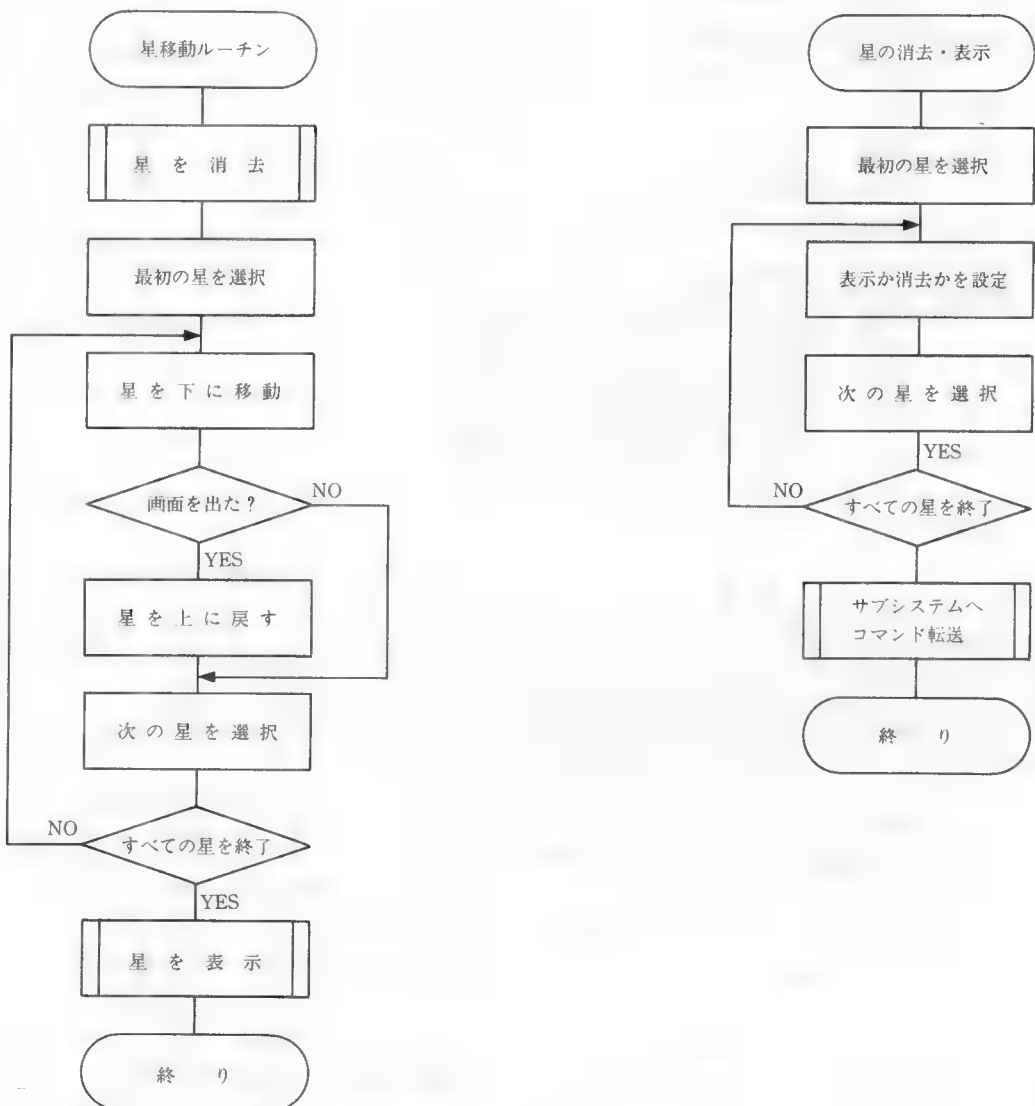
図D－14

行中に、更新または設定されます。

⑦では、次の星を選択し、⑧で20個すべての星について処理が終ったかどうかを調べています。終っていないのであればSTAR1から次の星の処理に戻ります。一方、全ての星について、処理が終っていれば、Aレジスタに0（ファンクションコードPSET）を代入して、星の消去・表示のルーチンへ入ります。

⑩以降がその星の消去・表示のルーチンです。まず⑩では、Xレジスタにサブシステムコマンドとして、サブCPUに渡そうとしているパラメータのある先頭アドレスをセットしています。このパラメータについては、前節で述べたものと同じです。

⑪では、Yレジスタに最初の星のファンクションコードが格納されているアドレスをセットし、カウンタとして用いるBレジスタに星の数20をセットしています。

⑫では、Aレジスタにパラメータとしてセットされているファンクションコード（表示＝0；消

図D－12　星移動ルーチンフローチャート

〔図D－13　STAR〕

```
*
* STAR MOVE
*
STAR    ①┌LDA     #1
         └BSR     STRPUT
STAR0   ②┌LDY     #20
         └LEAX    STRPTN+7,PCR
STAR1   ③┌LDD     ,X
         └ADDD    #4
        ④┌CMPD    #200
         └BLO     STAR2
        ⑤ SUBD    #200
STAR2   ⑥ STD     ,X
        ⑦ LEAX    6,X
        ⑧┌LEAY    -1,Y
         └BNE     STAR1
        ⑨ CLRA
STRPUT  ⑩ LEAX    STRPTN,PCR
        ⑪┌LEAY    10,X
         └LDB     #20
STRPT1  ⑫ STA     ,Y
        ⑬ LEAY    6,Y
        ⑭┌DECB
         └BNE     STRPT1
        ⑮ LBRA    TRANS
*
* STAR PATTERN
*
STRPTN  ⑯ FCB     STRP2-STRP1
STRP1   ⑰ FCB     0,0,$17,20
           FDB     046,000,$0104,310,100,$0204
           FDB     155,020,$0304,415,130,$0404
           FDB     235,040,$0504,512,150,$0604
           FDB     387,060,$0704,635,170,$0104
           FDB     464,080,$0204,024,190,$0304
           FDB     567,100,$0404,155,010,$0504
           FDB     625,120,$0604,278,030,$0704
           FDB     074,140,$0104,397,050,$0204
           FDB     187,160,$0304,488,070,$0404
           FDB     244,180,$0504,555,090,$0604
STRP2      EQU     *
```


去＝1）を、該当する所にセットしています。そして、⑬で次の星を選択し、⑭ですべての星が終るまで、STRPT1へループします。

最後に⑮で、ここでセットしたサブシステムコマンドをサブCPUに転送します。

それではここで、サブシステムコマンドをサブCPUに転送するルーチン、TRANSについて解説しておきましょう（フローチャートは省略します)。

図D—14がそのプログラムです。STOPとRUNのルーチンについては、第$C章3節で解説したものと、根本的には同じです。異なっているのは、$C章においては、メインCPUの割り込みを禁止したり許可したりしていましたが、このルーチンではこれを省略しています。というのは、このゲームプログラムでは、プログラムが走り始めた最初に割り込みを禁止して、このゲームの最中はずっと割り込みを禁止している状態にしてあるからです。この理由については後述します。

TRANSのルーチンは先に述べましたが、図D―10（前節）のようにパラメータを受け取って、サブCPUにサブシステムコマンドとして転送するサブルーチンです。

①で使用するレジスタをスタックへ退避します。そして、②でサブCPUを停止し、共有RAMをメインCPU側に接続します。

③では、共有RAMに転送するバイト数をパラメータから取り出して、④では転送先の共有RAMの先頭アドレスをUレジスタにセットしています。

⑤ではXレジスタの示すアドレスからUレジスタの示すアドレスへ1バイト転送し、⑥で、すべてを転送したかどうかをチェックし、まだ終了していなければループします。

そして、⑦でサブCPUの停止を解除し、

⑧で①でスタックへ退避したレジスタ群を復帰して、処理を終ります。

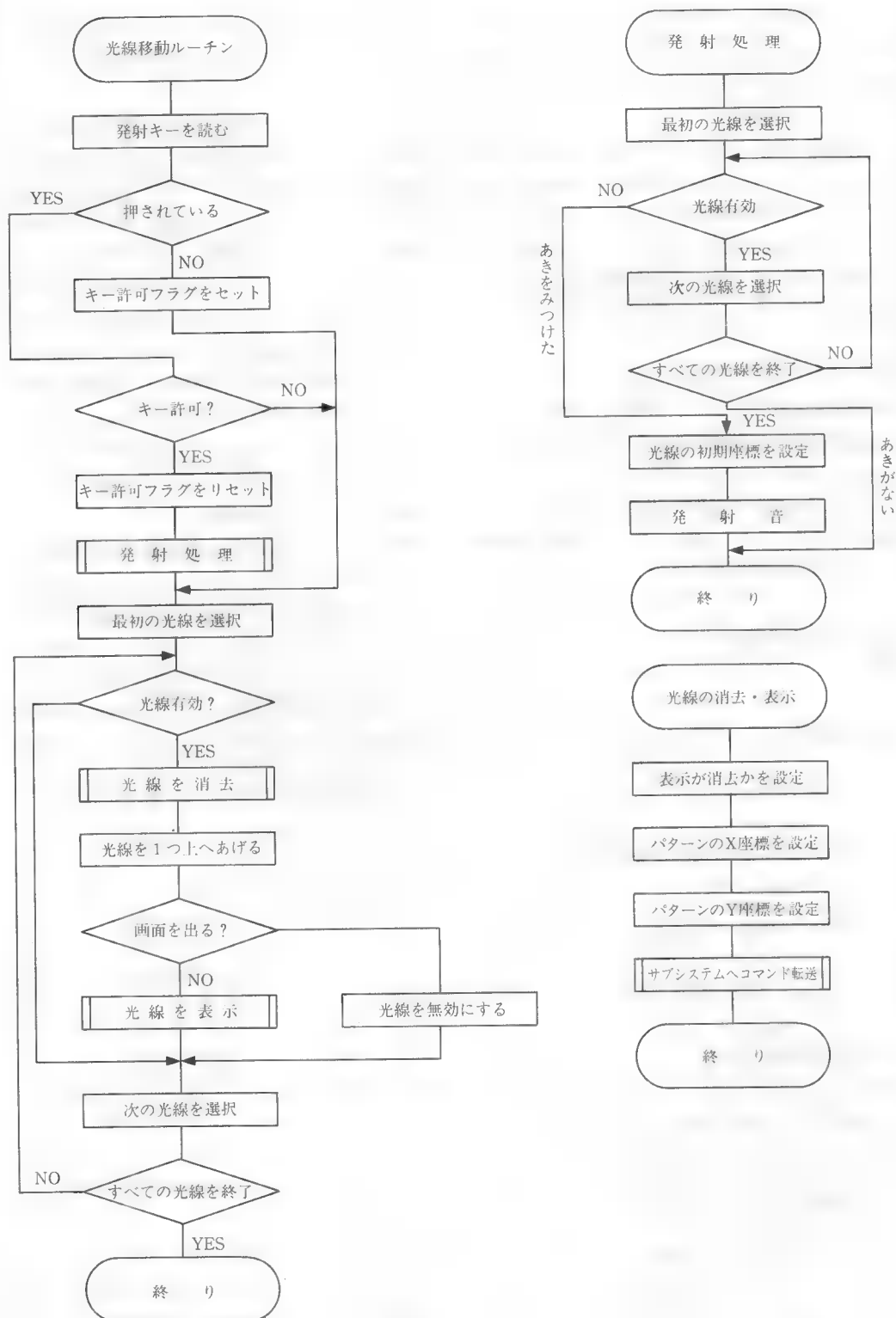
8. 光線移動ルーチン

次に光線移動ルーチンを考えます。このルーチンのディーテルフローチャートは図D―15です。このフローチャートに関して、若干解説しておきます。

FM―7シリーズでは、既に一度述べましたが、キーボードに（ゲームを作成するには）不便な点があります。というのは、キーが押されたことはわかっても、そのキーが**いつ離されたかを知ることはハードウェア上の制約により不可能である**ということです。ですから、FM―7シリーズ用のリアルタイムゲームなどでは、一度移動キーを押すとその方向に移動し続けるということになります。また、同じ理由で、**複数のキーが同時に押されていることを、知ることもできません**。この欠点は残念ながらソフトウェアだけではどうすることもできず、これを変えたければハードウェアの改造（または追加）が必要です。

でも実は、1つだけ以上の制約がない例外のキーがあります。それは、キーボード左上のBREAKキーです。このキーは、他のキーとは完全に独立しておりその読み出し方法なども全く異なっています。割込みのところで詳しく解説しますので、簡単な話だけしておくと、このBREAKキーが押されているか押されていないかは、$FD

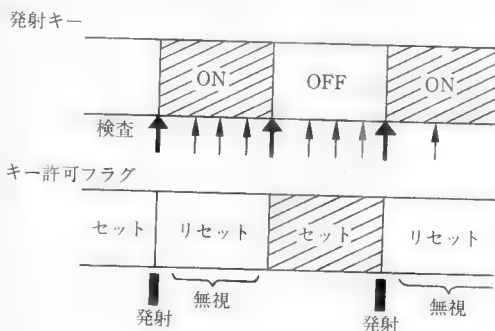
〔図D－14 STOP・RUN・TRANS〕

```
*
* SUBCPU CONTROL PROGRAMS
*
* STOP SUB CPU
STOP      PSHS   A       ┐
STOP1     LDA    HALT    │
          BMI    STOP1   │
          LDA    #$80    │……サブCPUを停止するルーチン
          STA    HALT    │
STOP2     LDA    HALT    │
          BPL    STOP2   │
          PULS   A,PC    ┘
* RUN SUB CPU
RUN       CLR    HALT    ┐
          RTS            ┘……サブCPUの停止を解除するルーチン
* TRANS
TRANS   ① PSHS   D,X,U
        ② BSR    STOP
        ③ LDB    ,X+
        ④ LDU    #SRAM
TRANS1  ⑤┌LDA    ,X+
         └STA    ,U+
        ⑥┌DECB
         └BNE    TRANS1
        ⑦ BSR    RUN
        ⑧ PULS   D,X,U,PC
```

図D−15　光線移動ルーチンフローチャート

04（BREAKというシンボルを付けます）番地のbit 1に示されています。このbitが0ならばBREAKキーが押されていることを、1ならば押されていないことを示しています。

このようにして、BREAKキーを利用すれば、通常のキーと違った使い方をすることができます。ただし、このBREAKキーを、この場合のように発射キーとして利用したりする場合には、メインCPUの割り込みを禁止しておかなければいけません（詳しい理由は割り込みのところで解説します）。

さて、このように、離していることのわかるキーを発射キーとして用いる場合には、ただ、キーが押されていたら発射するとすると、キーを押したままでもいくつも発射してしまいあまりおもしろくありません。そこで、このルーチンでは、一回押すと、1つ発射とするために、工夫します。

これは次のようにしています。キーが押されて、発射したら、キー許可フラグをリセットします。そして、次にこのルーチンへ来たときに発射キーがまだ押されていても、キー許可フラグがリセットされているときには、発射処理をしません。何回かこのルーチンを通った後、発射キーが押されていなかったら、キー許可フラグをセットします。そして、また何回か後にこのルーチンを通ったときに、発射キーが押されたとすると、この時にはキー許可フラグはセットされているので、発射処理をします。と同時に、キー許可フラグをリセットします。つまり、一度キーが離されて、キー許可フラグがセットされるまで次の発射処理を行わ

図D－16　キー許可フラグと発射キー

ないようにするわけです。（図D－16参照）

フローチャート内に出てくる光線の有効、無効というのは、（10コある光線のうち）その光線が有効なのか、つまり、画面上に存在するかどうかを示しています。

発射処理のルーチンでは、この10コある光線のうち無効の（つまりあいている）光線を捜して発射しているわけです。

それでは、実際のプログラムの説明に移ります。

図D－17　MSIL

```
*
* MISSILE MOVE ROUTINE
*
MSIL     EQU     *
      ①  LDA     BREAK
         ANDA    #$02
      ②  BEQ     MSIL1
      ③  STA     KEYFLG,PCR
         BRA     MSIL4
MSIL1 ④  TST     KEYFLG,PCR
         BEQ     MSIL4
      ⑤  CLR     KEYFLG,PCR
      ⑥  LDY     MISBUF,PCR
         LDB     MISILN,PCR
MSIL2 ⑦  TST     ,Y
         BEQ     MSIL3
      ⑧  LEAY    3,Y
         DECB
         BNE     MSIL2
         BRA     MSIL4
MSIL3 ⑨  LDB     #23
         LDA     CANX,PCR
         INCA
         STD     1,Y
         STB     ,Y
      ⑩  LDD     #$07F7
         LBSR    PSG
         LDD     #$0D09
         LBSR    PSG
MSIL4 ⑪  LDY     MISBUF,PCR
         LDB     MISILN,PCR
         PSHS    B
MSIL5 ⑫  TST     ,Y
         BEQ     MSIL6
      ⑬  LDA     #1
         BSR     MISPUT
      ⑭  LDB     2,Y
         DECB
         BMI     MSIL7
         STB     2,Y
      ⑮  CLRA
         BSR     MISPUT
MSIL6 ⑯  LEAY    3,Y
         DEC     ,S
         BNE     MSIL5
         PULS    B,PC
MSIL7 ⑰  CLR     ,Y
         BRA     MSIL6
```

(図D—17。このプログラム中では光線という用語ではなくミサイルということになっています。これは、開発時の気まぐれです。できあがってみると、ミサイルというより光線といった方がよいような気がしたのでこうしました)。

①では、発射キーを読んでいます。つまり、$FD04（BREAK）番地を読んで、次の

ANDA #$02

によりbit 1だけを取り出しています。

②では、キーが押されているかを判断しています。押されていれば、bit 1は0になるので、ANDをとると、ゼロになりZフラグがセットされていますので、MSIL1に分岐します。

③は、キーが押されていないときの処理で、キー許可フラグに0でない数を代入しています（Aレジスタは0ではありません)。そして、MSIL4に分岐します。

④からはキーが押されていたときの処理です。④ではキー許可フラグをテストして、キー許可フラグがリセット（0である）されているときには、発射処理をせずに、MSIL4へ分岐します。

⑤では、発射処理を行うに先だってキー許可フラグをリセットします。

⑥〜⑩が発射処理にあたります。ここではサブルーチンにはしませんでした。⑥では、Yレジスタに最初の光線の記憶領域の先頭アドレス（MISBUFに格納されている）をセット、Bレジスタに光線の個数（MISILNに格納されている。通常10が格納されている）をセットしています。

⑦では、光線の有効フラグを調べています。有効フラグは、0なら無効、0以外なら有効ということにします。無効なら、あきがみつかったのでMSIL3へ。

有効だったときは、次の光線について、同じことをします(⑧)。もし、すべての光線が有効だったときには、もう発射できないので、MSIL4へ。

⑨、⑩は、実際の発射を行います。まず⑨では、光線の初期座標（キャノンのすぐ上）を設定し、光線の有効フラグをセットします。そして⑩で光線発射の効果音を出します。ここでは、BASICでの、

SOUND 7,&HF7
SOUND &HD,&H09

に相当する処理を行っています。ここで用いているPSGというルーチンは、図D—18のようになっています。このPSGの使い方については、解析マニュアルフェーズIの233ページまたは、システム仕様の1—47ページを参照してください（ただし、FM—7のシステム仕様には、誤植、[誤$FD00→正$FD0D、誤$FD01→正$FD0E］があるので、注意すること)。

⑪からは、光線の移動を行うルーチンです。⑪では、最初の光線を選択すると同時にスタックトップ（スタックの一番上のこと。“，S”で使用できる値のこと）に、光線の数を設定し、カウンタとして用います。

⑫では、有効フラグをチェックして無効ならば移動する必要はないのでMSIL6へ分岐します。

⑬では、光線を消去しています。(MISPUTのルーチン（後述）を使用します)。

〔図D－18 PSG〕

```
*
* PSG CONTROL ROUTINE
*
PSG     PSHS   D ···························· 使用するレジスタを退避
        STA    PSGD ························· レジスタ番号を$FD0Eにセット
        LDA    #$03 ┐
        STA    PSGC ├······················· レジスタを選択するコマンド（$03）を実行する
        CLR    PSGC ┘
        STB    PSGD ························· レジスタに設定する値を$FD0Eセット
        DECA        ┐
        STA    PSGC ├······················· レジスタに値を設定するコマンド（$02）を実行する
        CLR    PSGC ┘
        PULS   D,PC ························· 終わり
```

⑭では、光線のＹ座標をデクリメントして、光線を上にあげています。ここで画面を出てしまったとき（Ｙ座標がマイナス）にはＭＳＩＬ７へ分岐して、その光線を無効にしてしまいます（⑰）。

そして⑮で、新しい位置に光線を表示します。

⑯では、次の光線を選択してループしています。もし、全ての光線について移動が終れば、このルーチンの処理は終了です。

図Ｄ―19は、光線の消去・表示を行うルーチンです。このルーチンを呼ぶときには、Ａレジスタにファンクションコード、Ｙレジスタに消去または表示したい光線の記憶領域の先頭アドレスを必要とします。

①ではファンクションコードを、②では、光線のＸ座標（左側と右側）、③では光線のＹ座標（上と下）を設定しています。⑤が、光線のパターンです。

〔図Ｄ－19　ＭＩＳＰＵＴ〕

```
MISPUT ┌LEAX   MISPTN,PCR
     ① └STA    13,X
       ┌LDA    1,Y
       │LDB    #8
       │MUL
     ② │STD    4,X
       │ADDD   #8-1
       └STD    8,X
       ┌LDA    2,Y
       │LDB    #8
     ③ │MUL
       │STD    6,X
       │ADDD   #8-1
       └STD    10,X
     ④ LBRA   TRANS
*
* MISSILE PATTERN
*
MISPTN ┌FCB    MISP2-MISP1
MISP1  │FCB    0,0,$1C
       │FDB    0,0,0,0
       │FCB    6,0,8
       │FCB    %00011000
       │FCB    %00011000
     ⑤ │FCB    %00011000
       │FCB    %00011000
       │FCB    %00011000
       │FCB    %00011000
       │FCB    %00011000
       │FCB    %00011000
MISP2  └EQU    *
```

9. 敵の移動の方法

さて、ここからがこのプログラムの山場です。

まず、各自でランダム（デタラメ）に動く敵の動かし方ということを少し考えてみてください。

完全にデタラメというのであれば、敵の移動の処理ごとに乱数で、８方向（縦横斜め）の移動方向を選んで、動かしてやればいいだろうと思います。でも、これではちょっと不都合が生じます。８方向を選択する確率が等しいと、いつまでも同じあたりをうろうろとするだけになってしまいます。８方向を選択する確率に変化をつければ同じあたりをうろうろということはなくなりますが、その動きは細かくデタラメに動くということにな

図Ｄ－20　敵移動用のデータの構造

り、ゲームとしてあまりパッとしなくなってしまいます（相手の動きが全く予想できないというのはあまり気分のいいものではありません。個人的意見ですが……)。

そこで、ここでは、いくつかの移動パターンを用意しておき、乱数でそれらを選んで移動するという方法を採用することにしました。用意するパターンは、基本的には８方向の移動ですが、複雑な移動パターンを作成することもできるようにします。このような方式を採用した結果、データ構造を図Ｄ—20のようにすることにしました。各移動パターンのデータは、Ｘ座標、Ｙ座標の変化分を２の補数で表現したものを並べておきます。(Ｘ、Ｙともに変化分０ならばデータの終りを示すことにする。図Ｄ—20移動方向データ参照)。そして、各パターンの先頭アドレスを移動パターンテーブルとしてもっています。

ですから移動パターンを決めるときには乱数によって、移動パターンを選択し、移動パターンテーブルによって移動方向データを得るわけです。

これにより、敵の一つ一つは、今あるＸ座標、Ｙ座標とそれに加えて、次に使用する移動方向データのあるアドレスを記憶しておく必要があります。ここのあたりのことは、文章よりも図Ｄ—20のほうがわかりやすいと思います。

10. 敵移動ルーチン

前述のようなデータ構造を反映して、作成したフローチャートが図Ｄ—21です。そしてそれにそって作成したプログラムが図Ｄ—22です。

①では、最初の敵を選択して、Ｙレジスタに敵位置記憶領域のアドレスをセットします。そして②で移動の処理を行い、③で次の敵を選択しループします。全ての敵が終れば、処理を終了します。

④からが一つの敵の移動を行うルーチンです。

④では、処理している敵が、光線にやられて、(他のルーチンによって）画面外に追い出されている場合に、EMY５に分岐します。

⑤で、敵を消去しています。⑥では、移動方向データをＤレジスタにロードしています。そして

〔図Ｄ−22　ENEMY〕

```
*
* ENEMY MOVE ROUTINE
*
ENEMY     EQU     *
      ① LDB     ENEMYN,PCR
         LDY     EMYBUF,PCR
         PSHS    B
ENEMY1 ② BSR     ENEMYX
      ③ LEAY    4,Y
         DEC     ,S
         BNE     ENEMY1
         PULS    B,PC
- - - - - - - - - - - - - - - - - - -
ENEMYX    EQU     *
      ④ TST     ,Y
         BMI     EMY5
      ⑤ LDA     #1
         LBSR    EMYPUT
EMY1  ⑥ LDD     [2,Y]
         BEQ     EMY6
      ⑦ ADDA    ,Y
         ADDB    1,Y
      ⑧ TSTA
         BMI     EMY5
         CMPA    #78
         BHI     EMY5
      ⑨ TSTB
         BMI     EMY5
         CMPB    #23
         BHI     EMY5
      ⑩ BEQ     EMY2
         CMPB    #22
         BNE     EMY4
EMY2  ⑬ PSHS    A
      ⑪ INCA
         CMPA    CANX,PCR
         BLO     EMY3
         SUBA    #3
         CMPA    CANX,PCR
         BHI     EMY3
      ⑫ LDA     #1
         STA     SIGNAL,PCR
EMY3     PULS    A
EMY4  ⑭ STD     ,Y
      ⑮ LDD     2,Y
         ADDD    #2
         STD     2,Y
      ⑯ CLRA
         BRA     EMYPUT

EMY5  ⑰ BSR     RND
      ⑱ ANDB    #64-1
         ADDB    #8
         CLRA
         EXG     A,B
      ⑲ STD     ,Y
EMY6  ⑳ BSR     RND
         LSRB
         LSRB
         LSRB
         ANDB    #PATNN-1
      ㉑ ASLB
      ㉒ LEAX    DTABLE,PCR ‥X←DTABLE
         LDD     B,X ………………… D←(X+B)
         LEAX    D,X ………………… X←X+D
      ㉓ STX     2,Y
         BRA     EMY1
```

敵移動主ルーチン
やられた敵？
敵 を 消 去
次の移動方向データを得る
移動方向データ終り
各座標を加えて移動
画面から出る
NO
キャノンにあたった
YES
キャノンがやられたフラグをセット
次の移動方向データにする
敵 を 表 示
終　り
初期座標決定
移動パターン選択

図D－21　敵移動ルーチンフローチャート

移動方向データが終り（0）であるときは、他の移動パターンを選択すべくEMY 6に分岐します。

⑦では、X、Y座標をそれぞれ移動方向データに加算して新しい位置を得ます。そして、⑧でX座標、⑨でY座標をチェックして、画面の外へ出たときには、EMY 5に分岐します。

⑩ではY座標が22か23の場合を選んでEMY 2の処理を行います。そのEMY 2では、⑪でキャノンと衝突していないかをチェックし、衝突していたら⑫で、キャノンがやられたことを示すフラグ（シグナル）をセットします。またこの部分ではAレジスタを破壊するので、⑬で退避しています。

最後に、新しい位置を敵位置記憶領域の該当するところにセットし(⑭)、同時に、移動方向データを示すポインタを次の移動方向データを示すようにセットしなおします(⑮)。そして、敵を表示して処理を終ります（⑯)。

EMY 5からは、新しく座標を設定する場合の処理です。⑰でBレジスタに乱数をえて、⑱で、X座標を8～71、Y座標を0とし、新しい座標をセットします。

そしてEMY 6からは新しい移動パターンを選択するルーチンです。まず⑳で0～(パターンの数－1）までの乱数をえて（よい乱数をえるためにチョット数を操作しています。)、㉑でそれを2倍し、㉒で移動パターンのアドレスをXレジスタに得て、該当する所にセットして（㉓）元へ戻ります。図D—23が、移動パターンのデータ部分です。ここでは、ポジションインディペンデントにするために、テーブルの値は相対値となっています。

図D—24は、乱数を発生するルーチンです。このルーチンでは、得られる乱数は、

$$X_{i+1} = (X_i \times RNDC + 13) \bmod 2^{16}$$

の下位8ビットを用いています。これは、混合型合同式法と呼ばれる乱数の生成法です(参考文献：RAM, 1978. 7月号P50／広済堂出版)。

計算法は図D—25を参照して解説すれば、①でb×d、②でc×b、③でa×dを計算して加算

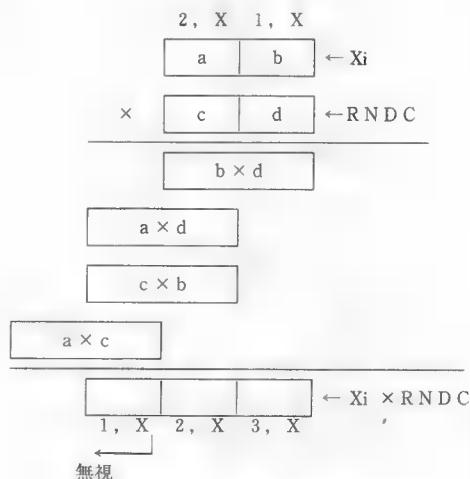
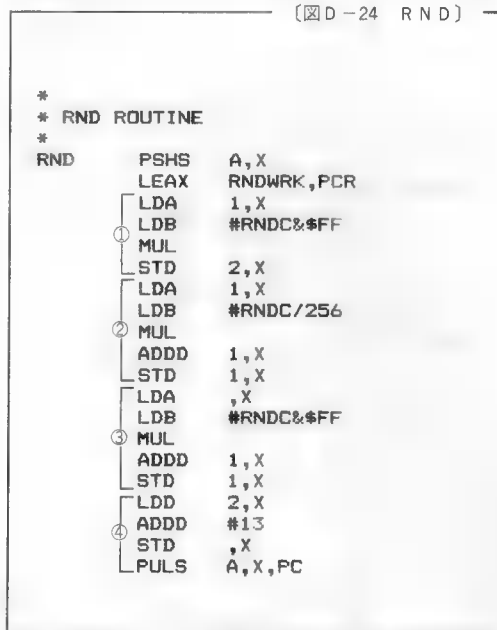
〔図D－23 DTABLE〕

```
*
* DATA OF ENEMY MOVE
*
PATNN   EQU     16
DTABLE  FDB     DELTA1-DTABLE
        FDB     DELTA2-DTABLE
        FDB     DELTA3-DTABLE
        FDB     DELTA4-DTABLE
        FDB     DELTA5-DTABLE
        FDB     DELTA6-DTABLE
        FDB     DELTA7-DTABLE
        FDB     DELTA8-DTABLE
        FDB     DELTA9-DTABLE
        FDB     DELTA0-DTABLE
        FDB     DELTA1-DTABLE
        FDB     DELTA2-DTABLE
        FDB     DELTA3-DTABLE
        FDB     DELTA4-DTABLE
        FDB     DELTA5-DTABLE
        FDB     DELTA6-DTABLE

DELTA1  FDB     $0001,$0001,$0001,$0001,0
DELTA2  FDB     $0002,$0002,$0002,$0002,0
DELTA3  FDB     $FF01,$FF01,$FF01,$FF01,0
DELTA4  FDB     $0101,$0101,$0101,$0101,0
DELTA5  FDB     $FF00,$FF00,$FF00,$FF00,0
DELTA6  FDB     $0100,$0100,$0100,$0100,0
DELTA7  FDB     $00FF,$00FF,$00FF,$00FF,0
DELTA8  FDB     $00FE,$00FE,$00FE,$00FE,0
DELTA9  FDB     $FFFF,$FFFF,$FFFF,$FFFF,0
DELTA0  FDB     $01FF,$01FF,$01FF,$01FF,0
```

しています。本来ならば、a×cを計算すべきですが、後に無視するので計算はしません。そして④で13を加算しています。

定数RNDCはうまく選ばないとよい乱数はえられません。ここでは、(他の所で定義しているのですが) 123を使用しています。123に固定するのであれば、c×bの計算は不用です（c＝0となるので）が汎用性を追求してそのままにしてあります。このRNDCを変えて実験してみるとおもしろいと思います。

〔図D－24 RND〕

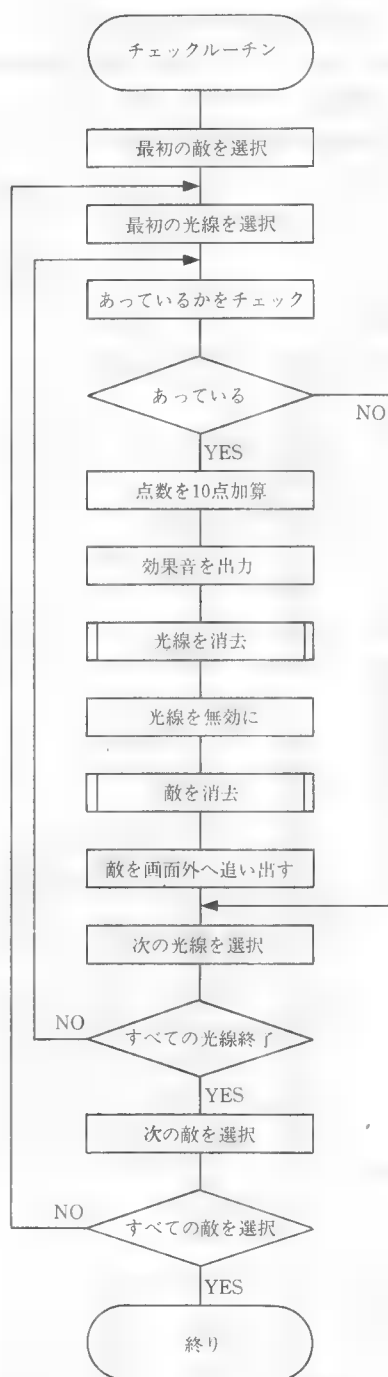
```
*
* RND ROUTINE
*
RND     PSHS    A,X
        LEAX    RNDWRK,PCR
      ┌ LDA     1,X
    ① │ LDB     #RNDC&$FF
      │ MUL
      └ STD     2,X
      ┌ LDA     1,X
      │ LDB     #RNDC/256
    ② │ MUL
      │ ADDD    1,X
      └ STD     1,X
      ┌ LDA     ,X
      │ LDB     #RNDC&$FF
    ③ │ MUL
      │ ADDD    1,X
      └ STD     1,X
      ┌ LDD     2,X
    ④ │ ADDD    #13
      │ STD     ,X
      └ PULS    A,X,PC
```

図D－25 乱数の発生

そして、最後に図D－26は敵を表示するルーチンです。このルーチンは、先に解説した、図D－19の光線表示のプログラムとほとんど同じですので解説は省略します。各自に解析してください。

〔図D－26 EMYPUT〕

```
*
* ENEMY DISPLAY
*
EMYPUT  LEAX    EMYPTN,PCR
        STA     13,X
        LDA     ,Y
        LDB     #8
        MUL
        STD     4,X
        ADDD    #2*8-1
        STD     8,X
        LDA     1,Y
        LDB     #8
        MUL
        STD     6,X
        ADDD    #2*8-1
        STD     10,X
        LBRA    TRANS
*
* ENEMY PATTERN
*
EMYPTN  FCB     EMYP2-EMYP1
EMYP1   FCB     0,0,$1C
        FDB     0,0,0,0
        FCB     3,0,8*4
        FCB     %11111111,%11111111
        FCB     %01110000,%00001110
        FCB     %00111000,%00011100
        FCB     %00011100,%00111000
        FCB     %00001110,%01110000
        FCB     %00000111,%11100000
        FCB     %00000011,%11000000
        FCB     %00111111,%11111100
        FCB     %00111111,%11111100
        FCB     %00000011,%11000000
        FCB     %00000111,%11100000
        FCB     %00001110,%01110000
        FCB     %00011100,%00111000
        FCB     %00111000,%00011100
        FCB     %01110000,%00001110
        FCB     %11111111,%11111111
EMYP2   EQU     *
```

11. チェックルーチン

次は、光線が敵にあたったかどうかをチェックするルーチンです。フローチャートは図D－27のようになります。このルーチンは、性質上光線移動のルーチンに含ませてもよいのですが、ここで

図D－27　チェックルーチンフローチャート

は独立させてサブルーチンとしました。 このルーチンのプログラムが図D－28です。ほとんどフローチャートに1対1に対応しているので、わかりやすいと思います。(もうこのあたりになると、詳しい説明はかえっておもしろくないでしょう。)

①、②がそれぞれ最初の敵、光線を選択する部分です。③が、光線が敵にあたったかどうかをチェックしている部分です。ここでは、敵の座標を（x, y）としたとき、(x, y)、(x＋1, y)、(x＋1, y＋1)、(x, y＋1) が光線の

〔図D－28　CHECK〕

```
*
* CHECK ROUTINE
*
CHECK   EQU     *
      ┌LDY     EMYBUF,PCR
     ①│LDB     ENEMYN,PCR
      └PSHS    B
CHECK1 ┌LDU     MISBUF,PCR
     ②│LDB     MISILN,PCR
      └PSHS    B
CHECK2 ┌LDD     ,Y
      │TST     ,U
      │BEQ     CHECK4
      │CMPD    1,U
      │BEQ     CHECK3
      │INCA
     ③│CMPD    1,U
      │BEQ     CHECK3
      │INCB
      │CMPD    1,U
      │BEQ     CHECK3
      │DECA
      │CMPD    1,U
      └BNE     CHECK4
CHECK3 ┌LDD     SCORE,PCR
      │ADDD    #10
     ④│STD     SCORE,PCR
      │LDD     #$07F9
      └LBSR    PSG
      ┌LDD     #$0D09
      │LBSR    PSG
      │EXG     Y,U
     ⑤│LDA     #1
      │LBSR    MISPUT
      │EXG     Y,U
      └CLR     ,U
      ┌LDA     #1
     ⑥│LBSR    EMYPUT
      │LDD     #$C0C0
      └STD     ,Y
CHECK4 ┌LEAU    3,U
     ⑦│DEC     ,S
      │BNE     CHECK2
      └PULS    B
      ┌LEAY    4,Y
     ⑧│DEC     ,S
      │BNE     CHECK1
      └PULS    B,PC
```

座標と一致しているかを調べています。

④〜⑥が当たったときの処理で、まず④では点数を更新し、効果音を出しています。⑤は、光線を消去する部分です。MISPUTのルーチンは、Yレジスタに光線の記憶領域の先頭アドレスを必要とするので、UレジスタとYレジスタとを入れ換えてサブルーチンを呼び出します。そして最後にその光線の有効フラグをクリアして無効にします。そして、⑥ではやられた敵を消去し、その敵を画面の外（X＝＄C０、Y＝＄C０）に追いだしておきます。

最後に、⑦、⑧はそれぞれ次の光線、敵を選択する部分です。

12. 初期化ルーチン

もうここまでくれば、あとは下り坂です。このルーチンは初期化の処理を順々にこなしていくだけなので、フローチャートは省略します。プログラムは図D—29になります。

①は、メインCPUの割り込みを禁止します。

〔図D－29 INIT〕

```
*
* INITIALIZE ROUTINE
*
INIT   ① ORCC   #$50
       ┌LEAX    SCRNI,PCR
      ②└LBSR    TRANS
       ③ BSR    INTMSG
INIT1  ┌LBSR    KEYIN
      ④ CMPA    #'          SPACE
       └BNE     INIT1
       ┌CLR     SIGNAL,PCR
      ⑤ LDD     #0
       └STD     SCORE,PCR
       ┌LEAX    BUFFER,PCR
       │STX     MISBUF,PCR
       │LDB     MISILN,PCR
INIT2 ⑥ CLR     ,X+
       │CLR     ,X+
       │CLR     ,X+
       │DECB
       └BNE     INIT2
       ┌STX     EMYBUF,PCR
       │LDB     ENEMYN,PCR
INIT3  │CLR     ,X
      ⑦ CLR     1,X
       │STX     2,X
       │LEAX    4,X
       │DECB
       └BNE     INIT3
       ┌LDA     #40
      ⑧└STA     CANX,PCR
       ┌LEAX    PSGDAT,PCR
       │LDB     #11
INIT4 ⑨ LDD     ,X++
       │LBSR    PSG
       │DECB
       └BNE     INIT4
       ┌LEAX    CLSCOM,PCR
      ⑩└LBRA    TRANS

SCRNI    FCB    11
         FCB    0,0,1,0,80,25,0,25,0,1,0
CLSCOM   FCB    7
         FCB    0,0,$0D,0,0,7
PSGDAT   FDB    $07F9,$0200,$0302,$0400
         FDB    $0504,$0600,$0810,$0910
         FDB    $0A10,$0B00,$0C10
```

この理由は後述します。②では、画面の初期設定（横80、縦25他）を行います。そして、③では、ゲームに関する説明文を出力します。

④ではスペースキーが押されるまで待ちます。

⑤では、キャノンがやられたことを示すフラグとスコアをクリアしています。

⑥では、光線記憶領域を順々に確保しています。まずMISBUFにその先頭アドレスをセットし光線の数×3バイト分をクリアします。

⑦では、光線記憶領域に続いて、敵位置記憶領域を確保しています。まず、EMYBUFにその先頭アドレスをセットし、敵の数×4バイト分を設定しています。この中の

STX 2,X

というのは、各敵の移動方向データのアドレスをその前の2バイト（CLR ,X　CLR 1,X）の0を指すようにしています。こうすること

〔図D－30　INTMSG〕

```
*
* OPENING MESSAGES OUTPUT
*
INTMSG   EQU     *
        ┌LDB     #3
        │LEAX    SYMCOM,PCR
INTMO   │PSHS    B
        │STB     12,X
        │ADDB    #160
       ①│STB     10,X
        │LBSR    TRANS
        │PULS    B
        │DECB
        └BNE     INTMO
        ┌DEC     12,X
        │DEC     10,X
        │LDA     #5
       ②│STA     4,X
        │LBSR    TRANS
        │LDA     #1
        └STA     4,X

        ┌CLRB
INTM1   │LBSR    PRINT
        │INCB
        │CMPB    #6
        │BNE     INTM1
        └RTS

SYMCOM   FCB     SCOM2-SCOM1
SCOM1    FCB     0,0,$19,1
         FCB     0,0,5,3
         FDB     160,0
         FCB     7
         FCC     /ZIG ZAP/
SCOM2    EQU     *
```

により、敵移動のルーチンで最初に移動パターンを選択します。

⑧では、キャノンの初期位置を画面中央（X＝40）にしています。

⑨では、PSGの各レジスタをPSGDATに従って初期設定しています。

そして⑩で、画面をクリアして、このルーチンの処理を終ります。

図D—30からは、この初期化ルーチンの下受けルーチンです。

図D—30は、ゲームの説明を出力するサブルーチンです。

①では、Bレジスタを3から1まで変化させて、BASICの

SYMBOL（160+B，B）……

に相当することを行います。

②では、色を換えて（青→水色）

SYMBOL（160，0）……

を行います。そして③では、PRINTサブルーチン（さらに下受けのサブルーチン）によってメッセージ番号0〜5までのメッセージを順々に出力します。

図D—31は、文字列を出力するサブルーチンです。メッセージの番号をBレジスタにいれて呼び出します。

出力するメッセージは図D—32のものです。各

〔図D－31　PRINT〕

```
*
* PRINT OUT MESSAGE
*
PRINT    PSHS    D,X,U
         LBSR    STOP ……………サブCPU停止
         LDU     #SRAM+2
         LDA     #$03…………PUTCコマンド
         STA     ,U+
         ASLB    ………………メッセージ番号を2倍に
         LEAX    MSGTBL,PCR…メッセージテーブルのアドレス
         LDD     B,X ……メッセージのオフセット
         LEAX    D,X ………メッセージのアドレス
         LDB     ,X ………………メッセージの長さ
PRINT1   LDA     ,X+    ┐
         STA     ,U+    │ メッセージを
         DECB           │ 共有RAMへ
         BPL     PRINT1 ┘
         LBSR    RUN ……………………コマンド実行

         PULS    D,X,U,PC
```

メッセージには、先頭にそのメッセージの長さをいれておきます。

図D―33は、キー入力を待つサブルーチンです。Aレジスタに入力されたキーのアスキーコードをいれて戻ります。ここでは、INKEYコマンドをキー入力を待つという指定で使っています。

〔図D－32　メッセージ〕

```
MSGTBL  FDB    MES0-MSGTBL
        FDB    MES1-MSGTBL
        FDB    MES2-MSGTBL
        FDB    MES3-MSGTBL
        FDB    MES4-MSGTBL
        FDB    MES5-MSGTBL
        FDB    MES6-MSGTBL

LOCATE  EQU    $12
CLS     EQU    $0C
MES0    FCB    MES1-MES0-1
        FCB    LOCATE,20,4
        FCC    "===== PART I  / Version 1.00 ====="
        FCB    LOCATE,8,6
        FCC    "ｺﾉ ｹﾞｰﾑ ﾊ ｢FM-7/FM-NEW7/FM-77 "
        FCC    "ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏﾆｭｱﾙ｣ ﾉ ﾀﾒﾉ"
        FCC    " ﾌﾟﾛｸﾞﾗﾑ ﾃﾞｽ｡"
MES1    FCB    MES2-MES1-1
        FCB    LOCATE,20,8
        FCC    "<< ｱ ｿ ﾋﾞ ｶ ﾀ >>"
        FCB    LOCATE,20,10
        FCC    " ｳｴｶﾗ ｵﾘﾃｸﾙ ｴｲﾘｱﾝ ｦ ｷｬﾉﾝ ﾉ "
        FCC    "ﾊｶｲ ｺｳｾﾝ ﾃﾞ ｹﾞｷﾊ ｼﾃ ｸﾀﾞｻｲ｡"
        FCB    LOCATE,20,12
MES2    FCB    MES3-MES2-1
        FCC    "<< ｷ ｰ ﾉ ｿ ｳ ｻ >>"
        FCB    LOCATE,18,13
        FCC    "    ┌─┐            "
        FCC    "             ┌─┐"
MES3    FCB    MES4-MES3-1
        FCB    LOCATE,18,14
        FCC    "<-- |4| LEFT MOVE  /"
        FCC    "  RIGHT MOVE |6| -->"
        FCB    LOCATE,18,15
        FCC    "    └─┘            "
        FCC    "             └─┘"
MES4    FCB    MES5-MES4-1
        FCB    LOCATE,18,16
        FCC    "   ┌─────┐      ┌"
        FCC    "──────┐"
        FCB    LOCATE,18,17
        FCC    "   |BREAK| FIRE    |"
        FCC    " SPACE | GAME START "
        FCB    LOCATE,18,18
        FCC    "   └─────┘      └"
        FCC    "──────┘"
MES5    FCB    MES6-MES5-1
        FCB    LOCATE,20,20
        FCC    "COPYRIGHT (C) 1984 "
        FCC    "by H.NAKAMURA"
        FCB    LOCATE,32,22
        FCC    "HIT SPACE KEY !"
MES6    FCB    MES7-MES6-1
        FCB    LOCATE,35,15
        FCC    "GAME  OVER"
        FCB    LOCATE,30,23
        FCC    "REPLAY ? (Y or N)"
        FCB    LOCATE,35,18
        FCC    "SCORE="
MES7    EQU    *
```

〔図D－33　GAME〕

```
*
* KEYIN ROUTINE
*
KEYIN    PSHS   B,X
         LEAX   INKEYC,PCR
         LBSR   TRANS ·························コマンド実行
         LBSR   STOP ··························サブCPU停止
         LDA    SRAM+3 ························キーコードの取出し
         LDB    SRAM       ┐
         ORB    #%10000000 │·················Ready Requstを設定
         STB    SRAM       ┘
         LBSR   RUN ···························サブCPU停止解除
         PULS   B,X,PC
INKEYC   FCB    EINKYC-SINKYC
SINKYC   FCB    0,0,$29,%00000011
EINKYC   EQU    *                ↑
                                 ·············キーバッファをクリアし，キー入力があるまで待つ
```

13. メインルーチン

さて最後になりました。いよいよメインルーチンの作成です。すでにフローチャートは、図D—3に示した通りです。また、プログラムは図D—34になります。

ここで、メインルーチンのループ中に、同じものがならべてあるわけを述べておきましょう。もしこうせずに、各ルーチンを1回ずつ呼び出すとすると、敵が1回動くごとにキャノンは1回しか動けず、(やってみるとわかりますが)非常にやりにくくなります。そこで、このように同じものを並べることにより、敵が1回動くごとに、光線とキャノンは2回動けるようにしたわけです。

図D—35は、点数を出力する下受けルーチンで

〔図D－34　GAME〕

```
*
* MAIN ROUTINE
*
GAME     EQU    *
         LBSR   INIT
MLOOP    LBSR   ENEMY
         LBSR   CANNON ┐
         LBSR   MSIL   │┐
         LBSR   CHECK  ┘│
         LBSR   CANNON ┐│···················同じものを並べている
         LBSR   MSIL   │┘
         LBSR   CHECK  ┘
         LBSR   STAR
         TST    SIGNAL,PCR ┐
         BEQ    MLOOP      ┘···············やられていなければループ
         LDB    #6     ┐
         LBSR   PRINT  ┘···················ゲームオーバーのメッセージを出力
         LBSR   SCOUT ·····················点数を出力
         LBSR   KEYIN  ┐
         CMPA   #'Y    │
         BEQ    GAME   │···················ReplayならGAMEへ
         CMPA   #'y    │
         BEQ    GAME   ┘
*
         ANDCC  #$AF ······················割込みを許可して
         RTS ······························終り
```

す。ここでは、点数を10000、1000、100、10、1で次々に割って各桁の数字を定めています。

図D—36に定数、図D—37に作業領域の定義部分を示しておきます。それぞれの役割りについては、既に各ルーチンで解説しました。

〔図D－37　作業領域〕

```
*
* MAIN ROUTINE WORK
*
SIGNAL  RMB    1
SCORE   RMB    2
CANX    RMB    1
MISBUF  RMB    2
EMYBUF  RMB    2
RNDWRK  FDB    1,0
KEYFLG  RMB    1
```

〔図D－36　定数〕

```
*
* EQUATES
*
HALT    EQU    $FD05
SRAM    EQU    $FC80
PSGC    EQU    $FD0D
PSGD    EQU    $FD0E
KEYBRD  EQU    $FD01
BREAK   EQU    $FD04

RNDC    EQU    123
*
* CONSTANT
*
ENEMYN  FCB    4
MISILN  FCB    10
```

14. ゲーム全リスト

以上で、このゲーム（ZIP　ZAP）の全ルーチンが完成しました。まとめとして、図D—38に全アセンブルリストをあけておきます。このアセンブルリストには、(本文中ではつけていなかった）注釈を数多くつけておきました。注釈のつけ方の参考にしてください（ただし、あまりいい例とはいえないかもしれません。とにかく自分にわかればいいので……)。既に述べましたが、このアセンブルリストは、FLEXというDOS上のT

〔図D－35　SCOUT〕

```
*
* SCORE OUTPUT
*
SCOUT   LEAU   NUMTB,PCR ............定数テーブルのアドレス
        LEAX   NUMBUF+5,PCR .........表示用のバッファ
        LDA    #5   ]
        PSHS   A    ]................5桁で出力
SCOUT1  LDA    #'0-1 ]
        PSHS   A     ]...............文字をセット
        LDD    SCORE,PCR ............スコアを取り出す
SCOUT2  INC    ,S ...................次の数字にする
        SUBD   ,U ...................引いてみて
        BCC    SCOUT2 ...............ひけたらさらに引き
        ADDD   ,U++ .................ひけなかったら，たして元に戻す
        STD    SCORE,PCR ............スコアを戻す
        PULS   A ....................得られた文字を
        STA    ,X+ ..................表示用バッファにセット
        DEC    ,S ...................カウンタをデクリメント
        BNE    SCOUT1 ...............ループする
        PULS   A ....................文字列完成
        LEAX   NUMBUF,PCR ]
        LBRA   TRANS      ]..........出力
NUMTB   FDB    10000,1000,100,10,1
NUMBUF  FCB    9
        FCB    0,0,$03,5
        FCC    /00000/
```

SC社のアセンブラによって出力したものです(つまりこのプログラムの開発はFLEX－DOS上で行ったというわけです)。しかし、このアセンブラ特有の擬似命令などはなるべく使用しないようにしました。

このリストを富士通のアブソリュートアセンブラを使用してアセンブルする際には、473行と482行の“&”(論理積)を“!.”に変更してください。その他のアセンブラを使用する際にも問題となるのは、2項演算子のところだけでしょう。いざとなれば、手で計算すればよいですから障害とはならないはずです。また、富士通のアセンブラを用いた場合ショートブランチでよいのにロングブランチを用いている箇所(アセンブルリスト中、左側に“>”が出力されているところ)で警告(WARNING)が出ますが、実行にはさしつかえありませんから無視してください。

アセンブルリストを打ちこんで実験するのがめんどうという人のために図D—39にダンプリストも示しました。ゲームだけを楽しみたいという人は、モニタなり(本書付録の)MIT7なりでこのダンプリストを入力してください。このダンプリストはMIT7で出力したもので、チェックサムが付いていますから、間違い捜しも容易でしょう。

それでは、マシン語の学習で疲れた頭をリフレッシュするのを兼ねて、このゲームで遊んでみてください。あくまでゲーム作りのサンプルなので、ゲームセンターのようには楽しめないかもしれませんが……。

〔図D－38 アセンブル・リスト〕

```
 1                          ******************************
 2                          *
 3                          * SAMPLE GAME PROGRAM
 4                          *        FOR
 5                          *    FM-7/NEW-7/77
 6                          *    ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏﾆｭｱﾙ
 7                          *
 8                          *  presented by H.NAKAMURA
 9                          *
10                          *  ` Z I G   Z A P '
11                          *      PART  I
12                          *       Ver 1.00
13                          *
14                          *  DATE 58/05/15 19:58
15                          *
16                          ******************************
17   2000                           ORG    $2000    program start from $2000
18   2000 20    02          START   BRA    GAME     branch to main program
19                          *
20                          * EQUATES
21                          *
22                  FD05    HALT    EQU    $FD05    I/O addr. of subCPU control
23                  FC80    SRAM    EQU    $FC80    top address of shared RAM
24                  FD0D    PSGC    EQU    $FD0D    PSG control register
25                  FD0E    PSGD    EQU    $FD0E    PSG data register
26                  FD01    KEYBRD  EQU    $FD01    keyboard address of mainCPU
27                  FD04    BREAK   EQU    $FD04    break key check address
28
29                  007B    RNDC    EQU    123      random number constant
30                          *
31                          * CONSTANT
32                          *
33   2002 04                ENEMYN  FCB    4        number of enemies
34   2003 0A                MISILN  FCB    10       number of missiles
35
36                          *
37                          * MAIN ROUTINE
38                          *
```

```
107    207A B7    FD0D             STA    PSGC       command set
108    207D 7F    FD0D             CLR    PSGC
109    2080 F7    FD0E             STB    PSGD       command clear
110    2083 4A                     DECA              data set command ($02)
111    2084 B7    FD0D             STA    PSGC       command set
112    2087 7F    FD0D             CLR    PSGC       command clear
113    208A 35    86               PULS   D,PC       return
114
115                       *
116                       * KEYIN ROUTINE
117                       *
118    208C 34    14      KEYIN    PSHS   B,X        save registers
119    208E 30    8D 0016          LEAX   INKEYC,PCR inkey command addr.
120   >2092 17    FFCA             LBSR   TRANS      transfer to subCPU
121   >2095 17    FFB0             LBSR   STOP       stop subCPU for data read
122    2098 B6    FCB3             LDA    SRAM+3     get key data
123    209B F6    FCB0             LDB    SRAM       request flag on
124    209E CA    80               ORB    #%10000000
125    20A0 F7    FC80             STB    SRAM
126   >20A3 17    FFB5             LBSR   RUN        restart subCPU
127    20A6 35    94               PULS   B,X,PC     return
128    20A8 04            INKEYC   FCB    EINKYC-SINKYC
129    20A9 00 00 29 03   SINKYC   FCB    0,0,$29,%00000011 INKEY with wait
130                 20AD  EINKYC   EQU    *
131
132                       *
133                       * PRINT OUT MESSAGE
134                       *
135    20AD 34    56      PRINT    PSHS   D,X,U      save registers
136   >20AF 17    FF96             LBSR   STOP       stop subCPU
137    20B2 CE    FC82             LDU    #SRAM+2    load destination
138    20B5 86    03               LDA    #$03       PUTC request
139    20B7 A7    C0               STA    ,U+
140    20B9 58                     ASLB              string no *2
141    20BA 30    8D 04E0          LEAX   MSGTBL,PCR table addr.
142    20BE EC    85               LDD    B,X        load offset
143    20C0 30    8B               LEAX   D,X        load message addr.
144    20C2 E6    84               LDB    ,X         load string length
145    20C4 A6    80      PRINT1   LDA    ,X+        transfer string
146    20C6 A7    C0               STA    ,U+
147    20C8 5A                     DECB
148    20C9 2A    F9               BPL    PRINT1
149   >20CB 17    FFBD             LBSR   RUN        restart subCPU
150    20CE 35    D6               PULS   D,X,U,PC   return
151
152                       *
153                       * SCORE OUTPUT
154                       *
155    20D0 33    8D 002D SCOUT    LEAU   NUMTB,PCR load convert table
156    20D4 30    8D 0038          LEAX   NUMBUF+5,PCR load buffer addr.
157    20D8 86    05               LDA    #5         length 5
158    20DA 34    02               PSHS   A
159    20DC 86    2F      SCOUT1   LDA    #'0-1      set chr.
160    20DE 34    02               PSHS   A
161    20E0 EC    8D FF58          LDD    SCORE,PCR load score
162    20E4 6C    E4      SCOUT2   INC    ,S         inc chr.
163    20E6 A3    C4               SUBD   ,U         score - 10^x
164    20E8 24    FA               BCC    SCOUT2
165    20EA E3    C1               ADDD   ,U++
166    20EC ED    8D FF4C          STD    SCORE,PCR set mod.
167    20F0 35    02               PULS   A          pull chr.
168    20F2 A7    80               STA    ,X+        set character
169    20F4 6A    E4               DEC    ,S         dec counter
170    20F6 26    E4               BNE    SCOUT1
171    20F8 35    02               PULS   A
172    20FA 30    8D 000D          LEAX   NUMBUF,PCR print out number
173    20FE 16    FF5E             LBRA   TRANS
```

```
174    2101 2710 03EB      NUMTB   FDB    10000,1000,100,10,1 convert table
       2105 0064 000A
       2109 0001
175    210B 09             NUMBUF  FCB    9
176    210C 00 00 03 05            FCB    0,0,$03,5
177    2110 30 30 30 30            FCC    /00000/   character buffer
       2114 30
178
179                        *
180                        * INITIALIZE ROUTINE
181                        *
182    2115 1A   50        INIT    ORCC   #$50      mask FIRQ/IRQ
183    2117 30   8D 005A           LEAX   SCRNI,PCR screen initialize
184    211B 17   FF41              LBSR   TRANS
185    211E 8D   7E                BSR    INTMSG    output opening mes.
186    2120 17   FF69      INIT1   LBSR   KEYIN     wait keyin
187    2123 81   20                CMPA   #'        SPACE
188    2125 26   F9                BNE    INIT1     if not space then loop
189    2127 6F   8D FF10           CLR    SIGNAL,PCR clear signal
190    212B CC   0000              LDD    #0        clear score
191    212E ED   8D FF0A           STD    SCORE,PCR
192    2132 30   8D 0688           LEAX   BUFFER,PCR set buffer addr.
193    2136 AF   8D FF05           STX    MISBUF,PCR missile buffer
194    213A E6   8D FEC5           LDB    MISILN,PCR reserve misbuf
195    213E 6F   80        INIT2   CLR    ,X+       3byte/1missile
196    2140 6F   80                CLR    ,X+
197    2142 6F   80                CLR    ,X+
198    2144 5A                     DECB
199    2145 26   F7                BNE    INIT2
200    2147 AF   8D FEF6           STX    EMYBUF,PCR set emybuf
201    214B E6   8D FEB3           LDB    ENEMYN,PCR initialize emybuf
202    214F 6F   84        INIT3   CLR    ,X        clear X,Y
203    2151 6F   01                CLR    1,X
204    2153 AF   02                STX    2,X       point to 0
205    2155 30   04                LEAX   4,X       next enemy
206    2157 5A                     DECB
207    2158 26   F5                BNE    INIT3
208    215A 86   28                LDA    #40       set cannon X
209    215C A7   8D FEDE           STA    CANX,PCR
210    2160 30   8D 0024           LEAX   PSGDAT,PCR PSG initialize
211    2164 C6   0B                LDB    #11       number of PSG data
212    2166 EC   81        INIT4   LDD    ,X++      get PSG data
213    2168 17   FF08              LBSR   PSG
214    216B 5A                     DECB
215    216C 26   F8                BNE    INIT4
216    216E 30   8D 000F           LEAX   CLSCOM,PCR clear screen
217    2172 16   FEEA              LBRA   TRANS
218
219    2175 0B             SCRNI   FCB    11        width 80,25 etc.
220    2176 00 00 01 00            FCB    0,0,1,0,80,25,0,25,0,1,0
       217A 50 19 00 19
       217E 00 01 00
221    2181 07             CLSCOM  FCB    7         clear screen
222    2182 00 00 0D 00            FCB    0,0,$0D,0,0,7
       2186 00 07
223    2188 07F9 0200      PSGDAT  FDB    $07F9,$0200,$0302,$0400
       218C 0302 0400
224    2190 0504 0600              FDB    $0504,$0600,$0810,$0910
       2194 0810 0910
225    2198 0A10 0B00              FDB    $0A10,$0B00,$0C10
       219C 0C10
226
227                        *
228                        * OPENING MESSAGES OUTPUT
229                        *
230              219E      INTMSG  EQU    *
231    219E C6   03                LDB    #3        for i=3 downto 1
232    21A0 30   8D 0029           LEAX   SYMCOM,PCR
```

```
366   22CB 30    8D 001D  MISPUT   LEAX   MISPTN,PCR load pattern
367   22CF A7    0D                STA    13,X       set function code
368   22D1 A6    21                LDA    1,Y        set Xpos
369   22D3 C6    08                LDB    #8
370   22D5 3D                      MUL
371   22D6 ED    04                STD    4,X
372   22D8 C3    0007              ADDD   #8-1
373   22DB ED    08                STD    8,X
374   22DD A6    22                LDA    2,Y        set Ypos
375   22DF C6    08                LDB    #8
376   22E1 3D                      MUL
377   22E2 ED    06                STD    6,X
378   22E4 C3    0007              ADDD   #8-1
379   22E7 ED    0A                STD    10,X
380   22E9 16    FD73              LBRA   TRANS      subCPU go
381
382                       *
383                       * MISSILE PATTERN
384                       *
385   22EC 16             MISPTN   FCB    MISP2-MISP1
386   22ED 00 00 1C       MISP1    FCB    0,0,$1C
387   22F0 0000 0000               FDB    0,0,0,0
      22F4 0000 0000
388   22F8 06 00 08                FCB    6,0,8
389   22FB 18                      FCB    %00011000
390   22FC 18                      FCB    %00011000
391   22FD 18                      FCB    %00011000
392   22FE 18                      FCB    %00011000
393   22FF 18                      FCB    %00011000
394   2300 18                      FCB    %00011000
395   2301 18                      FCB    %00011000
396   2302 18                      FCB    %00011000
397               2303    MISP2    EQU    *
398
399                       *
400                       * ENEMY MOVE ROUTINE
401                       *
402               2303    ENEMY    EQU    *
403   2303 E6    8D FCFB           LDB    ENEMYN,PCR select first enemy
404   2307 10AE 8D FD35            LDY    EMYBUF,PCR
405   230C 34    04                PSHS   B
406   230E 8D    08       ENEMY1   BSR    ENEMYX     each enemy move
407   2310 31    24                LEAY   4,Y        select next enemy
408   2312 6A    E4                DEC    ,S
409   2314 26    F8                BNE    ENEMY1
410   2316 35    84                PULS   B,PC
411
412               2318    ENEMYX   EQU    *
413   2318 6D    A4                TST    ,Y         dead enemy ?
414   231A 2B    47                BMI    EMY5
415   231C 86    01                LDA    #1         erase enemy
416   231E 17    0089              LBSR   EMYPUT
417   2321 EC    B8 02    EMY1     LDD    [2,Y]      load delta data
418   2324 27    48                BEQ    EMY6
419   2326 AB    A4                ADDA   ,Y         culc next position
420   2328 EB    21                ADDB   1,Y
421   232A 4D                      TSTA              test Xpos
422   232B 2B    36                BMI    EMY5
423   232D 81    4E                CMPA   #78
424   232F 22    32                BHI    EMY5
425   2331 5D                      TSTB              test Ypos
426   2332 2B    2F                BMI    EMY5
427   2334 C1    17                CMPB   #23
428   2336 22    2B                BHI    EMY5
429   2338 27    04                BEQ    EMY2
430   233A C1    16                CMPB   #22
431   233C 26    19                BNE    EMY4
```

```
 432   233E 34   02        EMY2    PSHS   A          Y=22 or 23
 433   2340 4C                     INCA
 434   2341 A1   8D FCF9           CMPA   CANX,PCR
 435   2345 25   0E                BLO    EMY3
 436   2347 80   03                SUBA   #3
 437   2349 A1   8D FCF1           CMPA   CANX,PCR
 438   234D 22   06                BHI    EMY3
 439   234F 86   01                LDA    #1         cannon dead
 440   2351 A7   8D FCE6           STA    SIGNAL,PCR
 441   2355 35   02        EMY3    PULS   A
 442   2357 ED   A4        EMY4    STD    ,Y         set new pos.
 443   2359 EC   22                LDD    2,Y        select new delta
 444   235B C3   0002              ADDD   #2
 445   235E ED   22                STD    2,Y
 446   2360 4F                     CLRA
 447   2361 20   47                BRA    EMYPUT
 448
 449   2363 8D   1D        EMY5    BSR    RND        random no. 8..71
 450   2365 C4   3F                ANDB   #64-1
 451   2367 CB   08                ADDB   #8
 452   2369 4F                     CLRA
 453   236A 1E   89                EXG    A,B
 454   236C ED   A4                STD    ,Y         set new position
 455   236E 8D   12        EMY6    BSR    RND        select new delta-pattern
 456   2370 54                     LSRB
 457   2371 54                     LSRB
 458   2372 54                     LSRB
 459   2373 C4   0F                ANDB   #PATNN-1
 460   2375 58                     ASLB
 461   2376 30   8D 01A0           LEAX   DTABLE,PCR load delta addr.
 462   237A EC   85                LDD    B,X
 463   237C 30   8B                LEAX   D,X
 464   237E AF   22                STX    2,Y
 465   2380 20   9F                BRA    EMY1
 466
 467                       *
 468                       * RND ROUTINE
 469                       *
 470   2382 34   12        RND     PSHS   A,X
 471   2384 30   8D FCBB           LEAX   RNDWRK,PCR Xi+1=Xi*RNDC+13(mod2^16)
 472   2388 A6   01                LDA    1,X
 473   238A C6   7B                LDB    #RNDC&$FF
 474   238C 3D                     MUL
 475   238D ED   02                STD    2,X
 476   238F A6   01                LDA    1,X
 477   2391 C6   00                LDB    #RNDC/256
 478   2393 3D                     MUL
 479   2394 E3   01                ADDD   1,X
 480   2396 ED   01                STD    1,X
 481   2398 A6   84                LDA    ,X
 482   239A C6   7B                LDB    #RNDC&$FF
 483   239C 3D                     MUL
 484   239D E3   01                ADDD   1,X
 485   239F ED   01                STD    1,X
 486   23A1 EC   02                LDD    2,X
 487   23A3 C3   000D              ADDD   #13
 488   23A6 ED   84                STD    ,X
 489   23A8 35   92                PULS   A,X,PC     return RND in Breg
 490
 491                       *
 492                       * ENEMY DISPLAY
 493                       *
 494   23AA 30   8D 001D   EMYPUT  LEAX   EMYPTN,PCR
 495   23AE A7   0D                STA    13,X       set function code
 496   23B0 A6   A4                LDA    ,Y         set grah Xpos
 497   23B2 C6   08                LDB    #8
 498   23B4 3D                     MUL
```

```
620    24F6 004A 008C                FDB    074,140,$0104,397,050,$0204
       24FA 0104 018D
       24FE 0032 0204
621    2502 00BB 00A0                FDB    187,160,$0304,488,070,$0404
       2506 0304 01E8
       250A 0046 0404
622    250E 00F4 00B4                FDB    244,180,$0504,555,090,$0604
       2512 0504 022B
       2516 005A 0604
623                   251A  STRP2    EQU    *
624
625                         *
626                         * DATA OF ENEMY MOVE
627                         *
628                   0010  PATNN    EQU    16
629    251A 0020            DTABLE   FDB    DELTA1-DTABLE
630    251C 002A                     FDB    DELTA2-DTABLE
631    251E 0034                     FDB    DELTA3-DTABLE
632    2520 003E                     FDB    DELTA4-DTABLE
633    2522 0048                     FDB    DELTA5-DTABLE
634    2524 0052                     FDB    DELTA6-DTABLE
635    2526 005C                     FDB    DELTA7-DTABLE
636    2528 0066                     FDB    DELTA8-DTABLE
637    252A 0070                     FDB    DELTA9-DTABLE
638    252C 007A                     FDB    DELTA0-DTABLE
639    252E 0020                     FDB    DELTA1-DTABLE
640    2530 002A                     FDB    DELTA2-DTABLE
641    2532 0034                     FDB    DELTA3-DTABLE
642    2534 003E                     FDB    DELTA4-DTABLE
643    2536 0048                     FDB    DELTA5-DTABLE
644    2538 0052                     FDB    DELTA6-DTABLE
645
646    253A 0001 0001       DELTA1   FDB    $0001,$0001,$0001,$0001,0
       253E 0001 0001
       2542 0000
647    2544 0002 0002       DELTA2   FDB    $0002,$0002,$0002,$0002,0
       2548 0002 0002
       254C 0000
648    254E FF01 FF01       DELTA3   FDB    $FF01,$FF01,$FF01,$FF01,0
       2552 FF01 FF01
       2556 0000
649    2558 0101 0101       DELTA4   FDB    $0101,$0101,$0101,$0101,0
       255C 0101 0101
       2560 0000
650    2562 FF00 FF00       DELTA5   FDB    $FF00,$FF00,$FF00,$FF00,0
       2566 FF00 FF00
       256A 0000
651    256C 0100 0100       DELTA6   FDB    $0100,$0100,$0100,$0100,0
       2570 0100 0100
       2574 0000
652    2576 00FF 00FF       DELTA7   FDB    $00FF,$00FF,$00FF,$00FF,0
       257A 00FF 00FF
       257E 0000
653    2580 00FE 00FE       DELTA8   FDB    $00FE,$00FE,$00FE,$00FE,0
       2584 00FE 00FE
       2588 0000
654    258A FFFF FFFF       DELTA9   FDB    $FFFF,$FFFF,$FFFF,$FFFF,0
       258E FFFF FFFF
       2592 0000
655    2594 01FF 01FF       DELTA0   FDB    $01FF,$01FF,$01FF,$01FF,0
       2598 01FF 01FF
       259C 0000
656
657    259E 000E            MSGTBL   FDB    MES0-MSGTBL
658    25A0 007A                     FDB    MES1-MSGTBL
659    25A2 00C9                     FDB    MES2-MSGTBL
660    25A4 0102                     FDB    MES3-MSGTBL
661    25A6 0155                     FDB    MES4-MSGTBL
```

```
 662   25A8 01BF                       FDB    MES5-MSGTBL
 663   25AA 01F5                       FDB    MES6-MSGTBL
 664
 665
 666                0012   LOCATE      EQU    $12
 667                000C   CLS         EQU    $0C
 668   25AC 6B             MES0        FCB    MES1-MES0-1
 669   25AD 12 14 04                   FCB    LOCATE,20,4
       25B0 3D 3D 3D 3D                FCC    "===== PART I  / Version 1.00 ====="
       25B4 3D 20 50 41
       25B8 52 54 20 49
       25BC 20 20 2F 20
       25C0 56 65 72 73
       25C4 69 6F 6E 20
       25C8 31 2E 30 30
       25CC 20 3D 3D 3D
       25D0 3D 3D
 670   25D2 12 08 06                   FCB    LOCATE,8,6
 671   25D5 BA C9 20 B9                FCC    "ｺﾉ ｹﾞｰﾑ ﾊ ｢FM-7/FM-NEW7/FM-77 "
       25D9 DE B0 D1 20
       25DD CA 20 A2 46
       25E1 4D 2D 37 2F
       25E5 46 4D 2D 4E
       25E9 45 57 37 2F
       25ED 46 4D 2D 37
       25F1 37 20
 672   25F3 CF BC DD BA                FCC    "ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏﾆｭｱﾙ｣ ﾉ ﾀﾒﾉ"
       25F7 DE 20 C6 AD
       25FB B3 D3 DD 20
       25FF CF C6 AD B1
       2603 D9 A3 20 C9
       2607 20 C0 D2 C9
 673   260B 20 CC DF DB                FCC    " ﾌﾟﾛｸﾞﾗﾑ ﾃﾞｽ｡"
       260F B8 DE D7 D1
       2613 20 C3 DE BD
       2617 A1
 674   2618 4E             MES1        FCB    MES2-MES1-1
 675   2619 12 14 08                   FCB    LOCATE,20,8
 676   261C 3C 3C 20 B1                FCC    "<< ｱ ｿ ﾋﾞ ｶ ﾀ >>"
       2620 20 BF 20 CB
       2624 DE 20 B6 20
       2628 C0 20 3E 3E
 677   262C 12 14 0A                   FCB    LOCATE,20,10
 678   262F 20 B3 B4 B6                FCC    " ｳｴｶﾗ ｵﾘﾃｸﾙ ｴｲﾘｱﾝ ｦ ｷｬﾉﾝ ﾉ "
       2633 D7 20 B5 D8
       2637 C3 B8 D9 20
       263B B4 B2 D8 B1
       263F DD 20 A6 20
       2643 B7 AC C9 DD
       2647 20 C9 20
 679   264A CA B6 B2 20                FCC    "ﾊｶｲ ｺｳｾﾝ ﾃﾞ ｹﾞｷﾊ ｼﾃ ｸﾀﾞｻｲ｡"
       264E BA B3 BE DD
       2652 20 C3 DE 20
       2656 B9 DE B7 CA
       265A 20 BC C3 20
       265E B8 C0 DE BB
       2662 B2 A1
 680   2664 12 14 0C                   FCB    LOCATE,20,12
 681   2667 38             MES2        FCB    MES3-MES2-1
 682   2668 3C 3C 20 B7                FCC    "<< ｷ ｰ ﾉ ｿ ｳ ｻ >>"
       266C 20 B0 20 C9
       2670 20 BF 20 B3
       2674 20 BB 20 3E
       2678 3E
 683   2679 12 12 0D                   FCB    LOCATE,18,13
 684   267C 20 20 20 20                FCC    "    ┌─┐                    "
       2680 98 95 99 20
       2684 20 20 20 20
       2688 20 20 20 20
```

〔図D－39　ダンプリスト〕

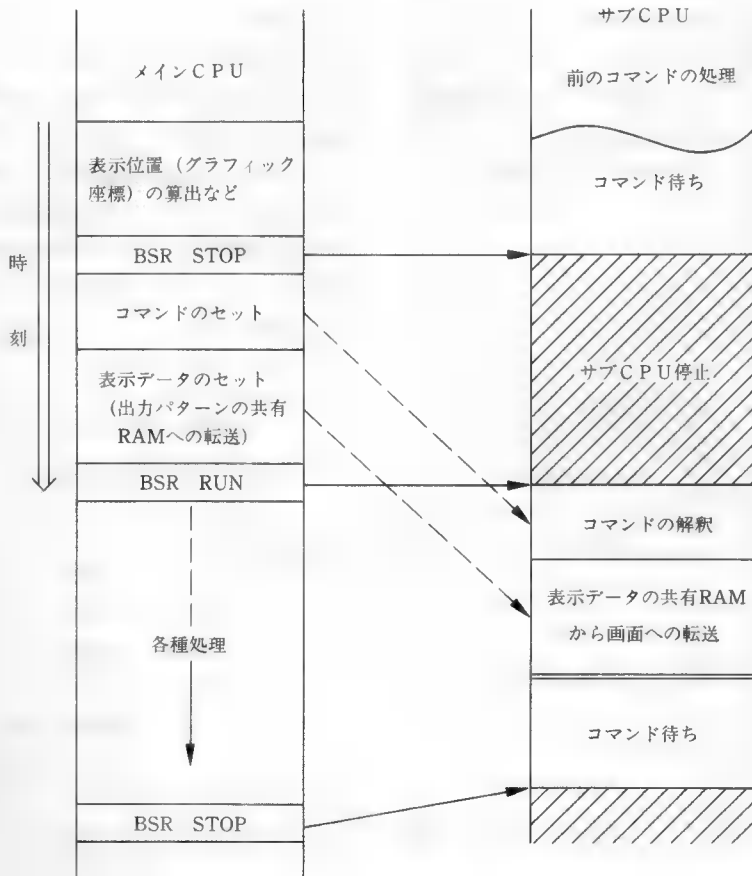
```
2000: 20 02 04 0A 17 01 0E 17 02 F9 17 01 D5 17 02 50 :BE  ........ホ..ユ..P
2010: 17 03 E7 17 01 CC 17 02 47 17 03 DE 17 04 49 6D :0E  ..◤..フ..G..゛..Im
2020: 8D 00 18 27 E2 C6 06 17 00 83 17 00 A3 17 00 5C :41  █..'ヰニ....」...¥
2030: 81 59 27 D0 81 79 27 CC 1C AF 39 A7 5E A7 5F 1A :E7  _Y'ミ_y'フ.ッ9ァ^ァ_.
2040: 01 39 34 00 01 00 00 84 34 02 B6 FD 05 2B FB 86 :8D  .94.....4.カ人.+町█
2050: 80 B7 FD 05 B6 FD 05 2A FB 35 82 7F FD 05 39 34 :BB  _キ人.カ人.*町5_.人.94
2060: 56 8D E5 E6 80 CE FC 80 A6 80 A7 C0 5A 26 F9 8D :0B  V█◣◥_ホ_ヲ_ァタZ&█
2070: EA 35 D6 34 06 B7 FD 0E 86 03 B7 FD 0D 7F FD 0D :C4  ◆5ヨ4.キ人.█.キ人..人.
2080: F7 FD 0E 4A B7 FD 0D 7F FD 0D 35 86 34 14 30 8D :56  ◈人.Jキ人..人.5█4.0█
2090: 00 16 17 FF CA 17 FF B0 B6 FC 83 F6 FC 80 CA 80 :AD  ... ハ. ーカ_分_ハ_
20A0: F7 FC 80 17 FF B5 35 94 04 00 00 29 03 34 56 17 :DB  ◈_. オ5....).4V.
20B0: FF 96 CE FC 82 86 03 A7 C0 58 30 8D 04 E0 EC 85 :3B  |ホ_█.ァタX0█.ニ█
20C0: 30 8B E6 84 A6 80 A7 C0 5A 2A F9 17 FF 8D 35 D6 :DD  0█◥_ヲ_ァタZ*ホ. █5ヨ
20D0: 33 8D 00 2D 30 8D 00 38 86 05 34 02 86 2F 34 02 :8E  3█.-0█.8█.4.█/4.
20E0: EC 8D FF 58 6C E4 A3 C4 24 FA E3 C1 ED 8D FF 4C :0E  ██ X1◢」ト$区ヨチ0█ L
20F0: 35 02 A7 80 6A E4 26 E4 35 02 30 8D 00 0D 16 FF :CC  5.ァ_j◢&◢5.0█...
2100: 5E 27 10 03 EB 00 64 00 0A 00 01 09 00 00 03 05 :00  ^'..♠.d.........
2110: 30 30 30 30 30 1A 50 30 8D 00 5A 17 FF 41 8D 7E :D3  00000.P0█.Z. A█~
2120: 17 FF 69 81 20 26 F9 6F 8D FF 10 CC 00 00 ED 8D :90  . i_ &ホo█ .フ..0█
2130: FF 0A 30 8D 06 88 AF 8D FF 05 E6 8D FE C5 6F 80 :B9  .0█.|ッ█ .◥█ナo_
2140: 6F 80 6F 80 5A 26 F7 AF 8D FE F6 E6 8D FE B3 6F :18  o_o_Z&◈ッ█▒分◥█▒ウo
2150: 84 6F 01 AF 02 30 04 5A 26 F5 86 28 A7 8D FE DE :0C  ▄o.ッ.0.Z&時█(ァ█▒゛
2160: 30 8D 00 24 C6 0B EC 81 17 FF 08 5A 26 F8 30 8D :72  0█.$ニ.●_. .Z&〒0█
2170: 00 0F 16 FE EA 0B 00 00 01 00 50 19 00 19 00 01 :9C  ...▒◆.....P.....
2180: 00 07 00 00 0D 00 00 07 07 F9 02 00 03 02 04 00 :26  .........ホ......
2190: 05 04 06 00 08 10 09 10 0A 10 0B 00 0C 10 C6 03 :4A  ..............ニ.
21A0: 30 8D 00 29 34 04 E7 0C CB A0 E7 0A 17 FE B0 35 :67  0█.)4.◤.ヒ ◤..▒-5
21B0: 04 5A 26 F0 6A 0C 6A 0A 86 05 A7 04 17 FE A0 86 :CF  .Z&×j.j.█.ァ..▒ █
21C0: 01 A7 04 5F 17 FE E6 5C C1 06 26 F8 39 14 00 00 :94  .ァ._.▒◥¥チ.&〒9...
21D0: 19 01 00 00 05 03 00 A0 00 00 07 5A 49 47 20 5A :2D  ....... ...ZIG Z
21E0: 41 50 86 01 8D 24 B6 FD 01 81 34 26 0C 6D 8D FE :5C  AP█.█$カ人._4&.m█▒
21F0: 4D 27 16 6A 8D FE 47 20 10 81 36 26 0C E6 8D FE :50  M'.j█▒G ._6&.◥█▒
2200: 3D C1 4D 24 04 6C 8D FE 35 4F 30 8D 00 13 A7 0D :72  =チM$.1█▒5O0█..ァ.
2210: A6 8D FE 2A C6 08 3D ED 04 C3 00 17 ED 08 16 FE :3A  ヲ█▒*ニ.=ㅇ.テ..ㅇ..▒
2220: 3E 3E 00 00 1C 00 00 00 B8 00 00 00 C7 05 00 30 :4C  >>......ク...ヌ..0
2230: 00 18 00 00 18 00 00 18 00 00 24 00 00 7E 00 18 :02  ..........$..~..
2240: 66 18 24 66 24 42 66 42 A5 E7 A5 A5 E7 A5 A5 FF :1C  f.$f$BfB・◤・・◤・・
2250: A5 99 00 99 A5 00 A5 C3 00 C3 81 00 81 81 00 81 :AB  ・▂.▂・.・テ.テ_._ _._
2260: B6 FD 04 84 02 27 06 A7 8D FD DC 20 35 6D 8D FD :C3  カ人.▄.'.ァ█人ワ 5m█人
2270: D6 27 2F 6F 8D FD D0 10 AE 8D FD C3 E6 8D FD 83 :F3  ヨ'/o█人ミ.ョ█人テ◥人_
2280: 6D A4 27 07 31 23 5A 26 F7 20 17 C6 17 A6 8D FD :4E  m、'.1#Z&◈ .ニ.ヲ█人
2290: AD 4C ED 21 E7 A4 CC 07 F7 17 FD D7 CC 0D 09 17 :40  ュLㅇ!◤、フ.◈.人ラフ...
22A0: FD D1 10 AE 8D FD 98 E6 8D FD 58 34 04 6D A4 27 :E6  人ム.ョ█人┌◥人X4.m、'
22B0: 0E 86 01 8D 16 E6 22 5A 2B 0D E7 22 4F 8D 0C 31 :F4  .█.█.◥"Z+.◤"O█.1
22C0: 23 6A E4 26 EB 35 84 6F A4 20 F4 30 8D 00 1D A7 :E0  #j◢&♠5▄o、 ㅂ0█..ァ
22D0: 0D A6 21 C6 08 3D ED 04 C3 00 07 ED 08 A6 22 C6 :1D  .ヲ!ニ.=ㅇ.テ..ㅇ.ヲ"ニ
22E0: 08 3D ED 06 C3 00 07 ED 0A 16 FD 73 16 00 00 1C :B1  .=ㅇ.テ..ㅇ..人s....
22F0: 00 00 00 00 00 00 00 00 06 00 08 18 18 18 18 18 :86  ................
2300: 18 18 18 E6 8D FC FB 10 AE 8D FD 35 34 04 8D 08 :FC  ...◥█_町.ョ█人54.█.
2310: 31 24 6A E4 26 F8 35 84 6D A4 2B 47 86 01 17 00 :9B  1$j◢&〒5▄m、+G█...
2320: 89 EC B8 02 27 48 AB A4 EB 21 4D 2B 36 81 4E 22 :98  ▌●7.'Hォ、♠!M+6_N"
2330: 32 5D 2B 2F C1 17 22 2B 27 04 C1 16 26 19 34 02 :85  2]+/チ."+'.チ.&.4.
2340: 4C A1 8D FC F9 25 0E 80 03 A1 8D FC F1 22 06 86 :EE  L。█_ホ%._.。█_円".█
2350: 01 A7 8D FC E6 35 02 ED A4 EC 22 C3 00 02 ED 22 :C1  .ァ█_◥5.ㅇ、●"テ..ㅇ"
2360: 4F 20 47 8D 1D C4 3F CB 08 4F 1E 89 ED A4 8D 12 :5C  O G█.ト?ヒ.O.▌ㅇ、█.
2370: 54 54 54 C4 0F 58 30 8D 01 A0 EC 85 30 8B AF 22 :82  TTTト.X0█. ●█0█ッ"
2380: 20 9F 34 12 30 8D FC BB A6 01 C6 7B 3D ED 02 A6 :33   ╱4.0█_サヲ.ニ{=ㅇ.ヲ
2390: 01 C6 00 3D E3 01 ED 01 A6 84 C6 7B 3D E3 01 ED :4F  .ニ.=ヨ.ㅇ.ヲ▄ニ{=ヨ.ㅇ
23A0: 01 EC 02 C3 00 0D ED 84 35 92 30 8D 00 1D A7 0D :85  .●.テ..ㅇ▄5┤0█..ァ.
23B0: A6 A4 C6 08 3D ED 04 C3 00 0F ED 08 A6 21 C6 08 :A2  ヲ、ニ.=ㅇ.テ..ㅇ.ヲ!ニ.
23C0: 3D ED 06 C3 00 0F ED 0A 16 FC 94 2E 00 00 1C 00 :E9  =ㅇ.テ..ㅇ.._┬.....
23D0: 00 00 00 00 00 00 00 00 03 00 20 FF FF 70 0E 38 1C :F3  ........,   p.8.
23E0: 1C 38 0E 70 07 E0 03 C0 3F FC 3F FC 03 C0 07 E0 :9C  .8.p.ニ.タ?_?_.タ.ニ
23F0: 0E 70 1C 38 38 1C 70 0E FF FF 10 AE 8D FC 42 E6 :11  .p.88.p.  .ョ█_B◥
2400: 8D FB FF 34 04 EE 8D FC 36 E6 8D FB F6 34 04 EC :F4  █町 4.╱█_6◥█町分4.●
```

```
2410: A4 6D C4 27 43 10 A3 41 27 12 4C 10 A3 41 27 0C :DF  、mト'C.」A'.L.」A'.
2420: 5C 10 A3 41 27 06 4A 10 A3 41 26 2C EC 8D FC 0C :8E  ¥.」A'.J.」A&,■■ホ.
2430: C3 00 0A ED 8D FC 05 CC 07 F9 17 FC 36 CC 0D 09 :3F  テ..C■ホ.フ.ホ.ホ6フ..
2440: 17 FC 30 1E 23 86 01 17 FE 81 1E 23 6F C4 86 01 :9C  .ホ0.#■..※_.#oト■.
2450: 17 FF 57 CC C0 C0 ED A4 33 43 6A E4 26 B1 35 04 :1E  . Wフタタ■、3Cj■&ア5.
2460: 31 24 6A E4 26 9F 35 B4 86 01 8D 1F 10 8E 00 14 :06  1$j■&'5■■.■..■..
2470: 30 8D 00 30 EC 84 C3 00 04 10 83 00 C8 25 03 83 :2A  0■.0■■テ...■.ネ%.■
2480: 00 C8 ED 84 30 06 31 3F 26 EA 4F 30 8D 00 0E 31 :3A  .ネ■■0.1?&◆O0■..1
2490: 0A C6 14 A7 A4 31 26 5A 26 F9 16 FB C2 7C 00 00 :4E  .ニ.ァ、1&Z&ホ.■ツ|..
24A0: 17 14 00 2E 00 00 01 04 01 36 00 64 02 04 00 9B :9A  ..........6.d...┘
24B0: 00 14 03 04 01 9F 00 82 04 04 00 EB 00 28 05 04 :61  ......■.■....♣.(..
24C0: 02 00 00 96 06 04 01 B3 00 3C 07 04 02 7B 00 AA :94  ...|....■.<...{.ェ
24D0: 01 04 01 D0 00 50 02 04 00 18 00 BE 03 04 02 37 :42  ...ミ.P......セ...7
24E0: 00 64 04 04 00 9B 00 0A 05 04 02 71 00 78 06 04 :0F  .d...┘......q.x..
24F0: 01 16 00 1E 07 04 00 4A 00 8C 01 04 01 8D 00 32 :DB  ........J.■....■.2
2500: 02 04 00 BB 00 A0 03 04 01 E8 00 46 04 04 00 F4 :93  ...サ. ....♠.F...B
2510: 00 B4 05 04 02 2B 00 5A 06 04 00 20 00 2A 00 34 :CC  .エ...+.Z... .*.4
2520: 00 3E 00 48 00 52 00 5C 00 66 00 70 00 7A 00 20 :A4  .>.H.R.¥.f.p.z.
2530: 00 2A 00 34 00 3E 00 48 00 52 00 01 00 01 00 01 :39  .*.4.>.H.R......
2540: 00 01 00 00 00 02 00 02 00 02 00 02 00 00 FF 01 :09  ...............
2550: FF 01 FF 01 FF 01 00 00 01 01 01 01 01 01 01 01 :08  ................
2560: 00 00 FF 00 FF 00 FF 00 FF 00 00 00 01 00 01 00 :FE  ................
2570: 01 00 01 00 00 00 00 FF 00 FF 00 FF 00 FF 00 00 :FE  ................
2580: 00 FE 00 FE 00 FE 00 FE 00 00 FF FF FF FF FF FF :F2  .※.※.※.※..
2590: FF FF 00 00 01 FF 01 FF 01 FF 01 FF 00 00 00 0E :0C  ................
25A0: 00 7A 00 C9 01 02 01 55 01 BF 01 F5 6B 12 14 04 :E7  .z./...U.ソ.■k...
25B0: 3D 3D 3D 3D 3D 20 50 41 52 54 20 49 20 20 2F 20 :80  ===== PART I  /
25C0: 56 65 72 73 69 6F 6E 20 31 2E 30 30 20 3D 3D 3D :9C  Version 1.00 ===
25D0: 3D 3D 12 08 06 BA C9 20 B9 DE B0 D1 20 CA 20 A2 :01  ==...コノ ゲーム ハ 「
25E0: 46 4D 2D 37 2F 46 4D 2D 4E 45 57 37 2F 46 4D 2D :F6  FM-7/FM-NEW7/FM-
25F0: 37 37 20 CF BC DD BA DE 20 C6 AD B3 D3 DD 20 CF :73  77 マシンゴ ニュウモン マ
2600: C6 AD B1 D9 A3 20 C9 20 C0 D2 C9 20 CC DF DB B8 :62  ニュアル」 ノ タメノ プロク
2610: DE D7 D1 20 C3 DE BD A1 4E 12 14 08 3C 3C 20 B1 :6A  ゛ラム デス。N...<< ア
2620: 20 BF 20 CB DE 20 B6 20 C0 20 3E 3E 12 14 0A 20 :4A   ソ ビ カ タ >>...
2630: B3 B4 B6 D7 20 B5 D8 C3 B8 D9 20 B4 B2 D8 B1 DD :E1  ウエカラ オリテクル エイリアン
2640: 20 A6 20 B7 AC C9 DD 20 C9 20 CA B6 B2 20 BA B3 :B7   ヲ キャノン ノ ハカイ コウ
2650: BE DD 20 C3 DE 20 B9 DE B7 CA 20 BC C3 20 B8 C0 :CB  セン デ ゲキハ シテ クタ
2660: DE BB B2 A1 12 14 0C 38 3C 3C 20 B7 20 B0 20 C9 :5E  ゛サイ。...8<< キ ー ノ
2670: 20 BF 20 B3 20 BB 20 3E 3E 12 12 0D 20 20 20 20 :DA   ソ ウ サ >>...
2680: 98 95 99 20 20 20 20 20 20 20 20 20 20 20 20 20 :66  ┌─┐
2690: 20 20 20 20 20 20 20 20 20 20 20 20 20 98 95 99 :66                 ┌─┐
26A0: 52 12 12 0E 3C 2D 2D 20 96 34 96 20 4C 45 46 54 :E5  R...<-- |4| LEFT
26B0: 20 4D 4F 56 45 20 20 2F 20 20 52 49 47 48 54 20 :A4   MOVE  /  RIGHT
26C0: 4D 4F 56 45 20 96 36 96 20 2D 2D 3E 12 12 0F 20 :C4  MOVE |6| -->...
26D0: 20 20 20 9A 95 9B 20 20 20 20 20 20 20 20 20 20 :6A     └─┘
26E0: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 :00
26F0: 9A 95 9B 69 12 12 10 20 20 20 98 95 95 95 95 95 :48  └─┘i...   ┌─────
2700: 99 20 20 20 20 20 20 20 20 20 98 95 95 95 95 95 :3A  ┐         ┌─────
2710: 95 95 99 12 12 11 20 20 20 96 42 52 45 41 4B 96 :E9  ──┐...   |BREAK|
2720: 20 46 49 52 45 20 20 20 20 96 20 53 50 41 43 45 :E8   FIRE    | SPACE
2730: 20 96 20 47 41 4D 45 20 53 54 41 52 54 20 12 12 :E2   | GAME START ..
2740: 12 20 20 20 9A 95 95 95 95 95 9B 20 20 20 20 20 :30  .   └─────┘
2750: 20 20 20 20 9A 95 95 95 95 95 95 95 9B 35 12 14 :23      └───────┘5..
2760: 14 43 4F 50 59 52 49 47 48 54 20 28 43 29 20 31 :D2  .COPYRIGHT (C) 1
2770: 39 38 34 20 62 79 20 48 2E 4E 41 4B 41 4D 55 52 :45  984 by H.NAKAMUR
2780: 41 12 20 16 48 49 54 20 53 50 41 43 45 20 4B 45 :AA  A. .HIT SPACE KE
2790: 59 20 21 2A 12 23 0F 47 41 4D 45 20 20 4F 56 45 :4C  Y !*.#.GAME  OVE
27A0: 52 12 1E 17 52 45 50 4C 41 59 20 3F 20 28 59 20 :86  R...REPLAY ? (Y
27B0: 6F 72 20 4E 29 12 23 12 53 43 4F 52 45 3D 00 00 :78  or N).#.SCORE=..
27C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :00  ................
27D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :00  ................
27E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :00  ................
27F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :00  ................
```

これはサブCPU上に表示データを置くということができない以上（取りあえず今の段階では）仕方のないことです。また、メインCPUがコマンドやパラメータをセットしている最中はサブCPUは停止してしまい何一つ仕事をなすことはないということです。

この点を考えると、高速化（というよりも非低速化といった方がそのイメージに近いかもしれません。）するためには、表示データを少くして、転送に消費する時間を節約し、サブCPUの停止を少しでも減らすということが望まれます。しかし、表示データを減らすというためには、画面のパターンを小さくするか、単色化するかのどちらかしかありません。画面のパターンを小さくするのは限界がある以上、単色化するしかありません（実際、前章のプログラムではパターンは単一色に限っています)。各パターンに細い色をつけられないというのは、おもしろいゲームとしては、失格の要因となりえる重大な事項であるのでこの点は、改善の余地を求めたいところでしょう。

2. 高速化するには

前節で述べたような欠点は、サブCPUに表示データや専用のプログラムを送ることさえできれ

図E－1　PGBLK1の場合

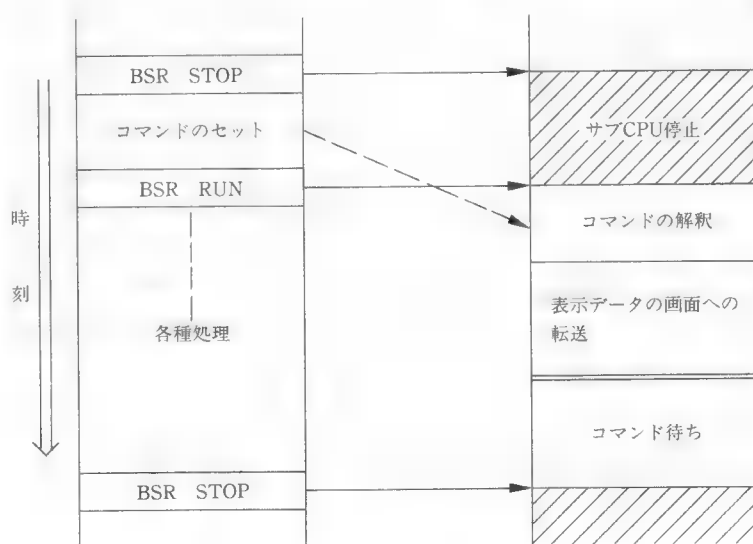
ば、解決します。つまり図E―1における、メインCPU側での表示データのセットを表示のつど行うのではなく、あらかじめサブCPU側に表示データを転送してサブCPU側のどこか適当なところに格納しておき、表示の際には、そのサブCPU側に格納されたデータを用いることによって、メインCPUからの表示データのセットを省略しようというわけです（図E―2）。

そうすると、画面への表示をする際には、メインCPUは簡単なコマンドのセットだけを行えばよく、表示データのセットは行わなくてよいので、その分だけサブCPUが停止している時間が短くなり高速化されます。

また、図E―1ではメインCPUで行っていた表示位置の算出（キャラクタ座標からグラフィック座標への変換）をやめて、サブCPUへはキャラクタ座標で表示位置を伝えてやり、グラフィック座標への変換はサブCPU側でやるようにするというのも高速化の一つの手段です（この座標変換に関しては、サブCPUでの処理を考えれば、サブCPUへはキャラクタ座標で表示位置を与えた方が、処理が少なくてすむという利点もあります）。

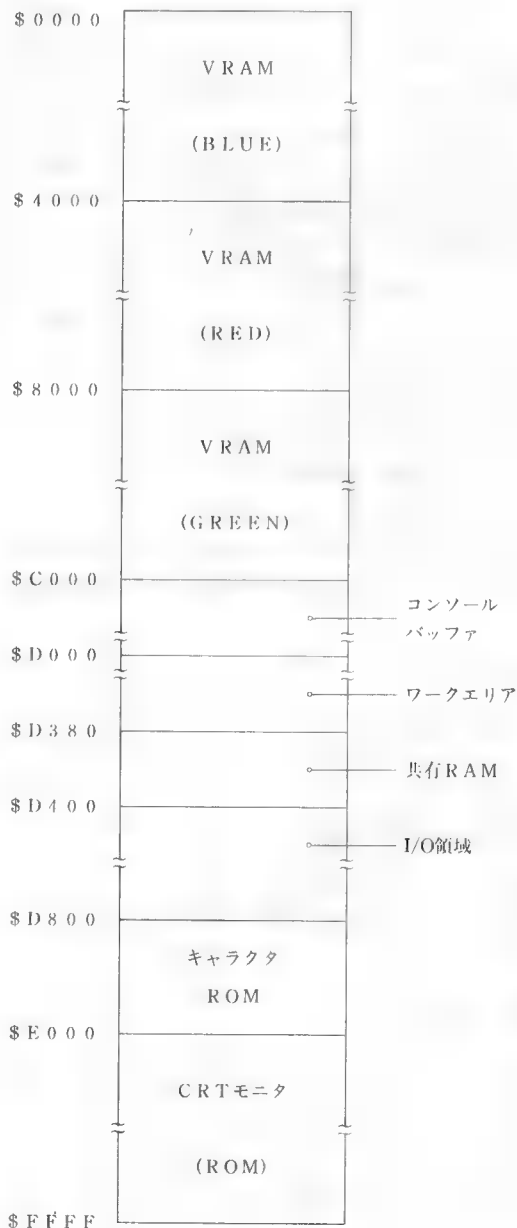
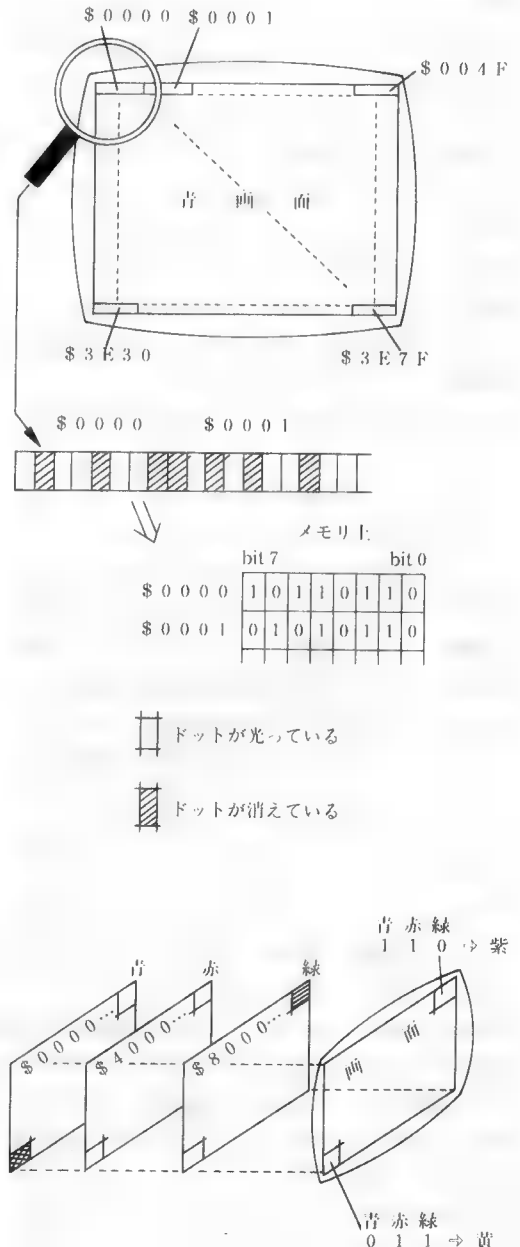
このように、表示データや専用のプログラムをサブCPUに送ることができれば、大幅な処理速度の効上も夢ではないことがわかります。普通ではメインCPUとサブCPUの仕事の量は、作成するプログラムによって違いがあるので、仕事の量が少ない方は、(図E―1や図E―2のサブCPUのコマンド待ちのように）仕事の多い方の仕事の終了を待ってただ時間を費すということが生じてしまいます（これが2つのCPUで、1つのCPUのときの2倍の処理をこなすことを期待できない一つの理由でもあります）。しかしもし、サブCPUにプログラムを送ってやることができて、その際、メインCPUとサブCPUの仕事の量をなるべく均等になるように分担すれば、仕事の終了待ちなどのむだな時間を少くすることができて処理速度の効上を期待することもできるというものです。

さて、それでは、この「サブCPUに表示データや専用のプログラムを送る」ということはできるのでしょうか。結論から言えば可能です。この方法についてはFM―7付属のマニュアル類には触れられていませんが、既に雑誌や、解析マニュアルなどで詳しく解析され実用に供されています。

そこで次節からは、その方法を解説をしていくと同時に、これを前章で作成したゲームに応用しさらに高速化をめざしていくことにしましょう。

図E－2　サブCPUへデータを転送しておいた場合

3. TESTコマンド

まず、サブCPUへのデータやプログラムの転送を行う命令を解説する前にサブCPUのメモリマップとその構造について説明を加えておきます。

図E—3がサブCPUのメモリマップです。$0000～$BFFFはVRAMです。この部分は画面のドットと対応しています。例えば画面をクリアするということは、$0000～$BFFFを0で埋めるということですし、$4000番地に、%10000000＝$80を書き込むと画面右上に

図E－3 サブCPUメモリマップ

図E－4 画面とVRAM

赤の点が1つ表示されます。このVRAMと画面の対応は図E―4に示すようになっています。この図では青の画面について書いてありますが、実際には、この青と赤、緑の画面が重なって表示されており、各画面のドットの明滅によって8色の色が表現されます。

＄C000～＄CFFFはコンソールバッファです。ここには、キャラクタとして表示した文字が画面のどの位置にあるかや、その色などを記憶しておくところです。

＄D380～＄D3FFが共有RAMです。メイン側では共有RAMは＄FC80～に配置されていましたが、サブ側ではこのアドレスになっています。ですから、メイン側で＄FC82にコマンドをセットするとサブ側では＄D382で読み取れるわけです。サブCPUは＄D380からにメインCPUでセットされるコマンドを読み取って各種のコマンドを実行するわけです。

＄D400～＄D7FFは、サブCPUでのI／O領域です。

＄D800～＄DFFFには、画面に表示される文字の表示パターンが格納されています。

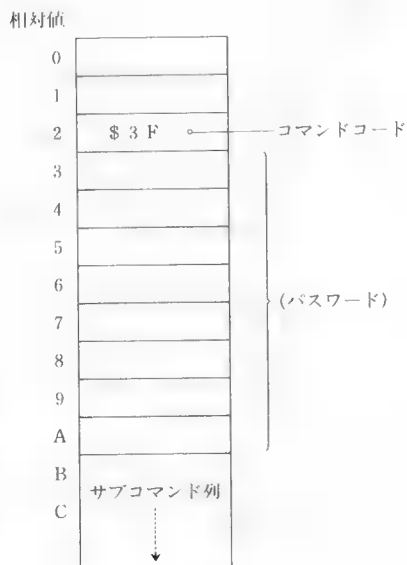
そして＄E000～＄FFFFがCRTモニタと呼ばれるものです。これには、サブCPUの処理（各コマンドの処理やキーボード、タイマの管理など）を行うプログラムが格納されています。このROMの内容については、解析マニュアル　フェーズIIIに詳しく解説されていますので興味のある方はそちらを参照してください。このROM上のプログラムは＄D000～のワークエリアを使用します。

それでは実際にデータやプログラムを送ってやる方法を解説しましょう。これを行うにはサブシステムコマンドのTESTコマンドを用います。TESTコマンドのパラメータを図E―5にあげました。相対値3～＄AはFM―8時代の名残りで、FM―8では‘YAMAUCHI’のアスキーコード（すなわち、＄59、＄41……）をセットしなければなりませんでした。このためこのTESTコマンドは別名「YAMAUCHI（ヤマウチ）コマンド」とも呼ばれています。FM―7シリーズでは、このパスワードは不用で、この部分には任意の値を設定できます（パスワードの照合はありませんが8文字分の領域だけは必要です）。

相対値＄Bからは、サブコマンド列となっています。このTESTコマンドには、4つのサブコマンドがあり、このサブコマンドをこの相対値＄Bから順々に並べておきます。サブコマンドには、図E―6に示す4種類があります。

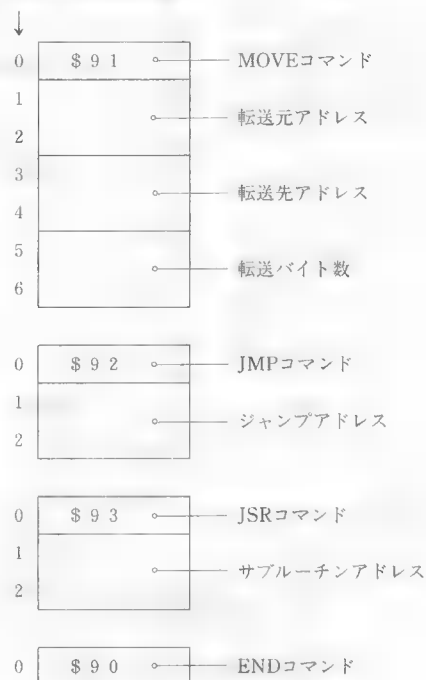
まず、MOVEコマンドはサブCPU側でデータなどの転送を行うサブコマンドです。例えば共有RAM上のデータをサブCPUのメモリのどこかに転送したい場合などに用います。ここで指定する転送元などのアドレス値は、サブCPU側でのアドレスを用いる必要があります。

JMPコマンドは、サブコマンドの制御を移すものです。例えば、＄92、＄C0、＄00とすれば次のサブコマンドは＄C000番地から取り出されて実行されます。このサブコマンドはあまり用いられません。

JSRコマンドは、その名前からJMPコマンドと混同される場合があるのですが、JMPコマンドとは全く異なっており、このサブコマンドがこのTESTコマンドの大黒柱ともいえる重要な存在です。このサブコマンドは、サブCPU側に

図E－5　TESTコマンド

置かれたマシン語サブルーチンをサブＣＰＵで実行するためのコマンドです。実行させるプログラムは、ＣＲＴモニタ内にあるサブルーチンや、メイン側からサブ側に送ったプログラムで、かつ、プログラムの最後が、 RTS 命令（または同等の PULS PC など）で終っているものでなければいけません。しかし、この条件を満たしていれば、プログラムの内容はどのようなものであってもよい（ただし厳密には、いろいろな制限もありますが）ので、例えばサブＣＰＵ上のＣＲＴモニタを離れた、全く独立したプログラムを走らせることも可能です。

最後にＥＮＤコマンドはサブシステムにＴＥＳＴコマンドのサブコマンド列の終りを示すものでサブコマンド列の最後には、必ずこのＥＮＤコマンドを付加しなければいけません。

それでは実際の例で見ましょう。図Ｅ―7がサンプルプログラムです。まず17行めから44行めまでは、すでに何度も使用している（前章のゲームの中で用いた物と同じ）サブＣＰＵを制御するプログラムです。

図E－6　サブコマンド

〔図Ｅ－7　ＴＥＳＴコマンドの使用例〕

```
 1                        *
 2                        * TEST PROGRAM OF 'TEST COMMAND
 3                        *
 4  5000                           ORG     $5000
 5                  5000  START    EQU     *
 6                        *
 7                        * EQUATES
 8                        *
 9                  FD05  HALT     EQU     $FD05
10                  FC80  SRAM     EQU     $FC80
11                        *
12  5000 30    8D 0038              LEAX    CUROFF,PCR  ]……カーソル消去
13  5004 8D    22                   BSR     TRANS       ]
14  5006 30    8D 0037              LEAX    SUBCPU,PCR  ]……TESTコマンドの実行
15  500A 8D    1C                   BSR     TRANS       ]
16  500C 39                         RTS
17                        *
18                        * SUBCPU CONTROL PROGRAMS
19                        *
20                        * STOP SUB CPU
21  500D 34    02         STOP     PSHS    A
22  500F B6    FD05       STOP1    LDA     HALT
23  5012 2B    FB                  BMI     STOP1
24  5014 1A    50                  ORCC    #$50
25  5016 86    80                  LDA     #$80
26  5018 B7    FD05                STA     HALT
27  501B B6    FD05       STOP2    LDA     HALT
```

```
 28   501E 2A   FB                BPL     STOP2
 29   5020 35   82                PULS    A,PC
 30                     * RUN SUB CPU
 31   5022 7F   FD05    RUN       CLR     HALT
 32   5025 1C   AF                ANDCC   #$AF
 33   5027 39                     RTS
 34                     * TRANS
 35   5028 34   56      TRANS     PSHS    D,X,U
 36   502A 8D   E1                BSR     STOP
 37   502C E6   80                LDB     ,X+
 38   502E CE   FC80              LDU     #SRAM
 39   5031 A6   80      TRANS1    LDA     ,X+
 40   5033 A7   C0                STA     ,U+
 41   5035 5A                     DECB
 42   5036 26   F9                BNE     TRANS1
 43   5038 8D   E8                BSR     RUN
 44   503A 35   D6                PULS    D,X,U,PC
 45                     *
 46   503C 04           CUROFF    FCB     COFFE-COFFS
 47   503D 00 00 0C 1E  COFFS     FCB     0,0,$0C,%011110
 48               5041  COFFE     EQU     *
 49                     *
 50               D40D  INSLED    EQU     $D40D
 51                     *
 52   5041 29           SUBCPU    FCB     SCPUE-SCPUS
 53                     *
 54   5042 00 00 3F     SCPUS     FCB     0,0,$3F
 55   5045 46 4D 37 CA            FCC     /FM7ﾊｽｺﾞｲ/
      5049 BD BA DE B2
 56   504D 93                     FCB     $93
 57   504E D38F                   FDB     $D38F
 58   5050 90                     FCB     $90
 59                     *
 60   5051 B6   D40D              LDA     INSLED
 61   5054 C6   05                LDB     #5
 62   5056 B6   D403    LOOP      LDA     $D403
 63   5059 8E   0000              LDX     #0
 64   505C 30   01      LOOP1     LEAX    1,X
 65   505E 26   FC                BNE     LOOP1
 66   5060 30   01      LOOP2     LEAX    1,X
 67   5062 26   FC                BNE     LOOP2
 68   5064 5A                     DECB
 69   5065 26   EF                BNE     LOOP
 70   5067 B7   D40D              STA     INSLED
 71   506A 39                     RTS
 72               506B  SCPUE     EQU     *
 73
 74                               END     START

0 ERROR(S) DETECTED

SYMBOL TABLE:

COFFE   5041   COFFS   503D   CUROFF 503C   HALT    FD05   INSLED D40D
LOOP    5056   LOOP1   505C   LOOP2  5060   RUN     5022   SCPUE  506B
SCPUS   5042   SRAM    FC80   START  5000   STOP    500D   STOP1  500F
STOP2   501B   SUBCPU  5041   TRANS  5028   TRANS1  5031
```


12行目からがメインルーチンです。ここではまずＣＵＲＯＦＦ（46～48行）によりカーソルを消しています。なぜ、このようにするかについての詳しいことはフェーズⅢに述べてあるので省略しますが、これをしないと動作がおかしくなる場合があります。そして、次にＳＵＢＣＰＵ(52行目)から後の部分を共有ＲＡＭに転送して処理を終ります。メインＣＰＵは、共有ＲＡＭへデータをセットしているだけです。

それではＳＵＢＣＰＵから後の部分を詳しく解説しましょう。この部分は、図Ｅ—8のように共有ＲＡＭへ転送されます。＄ＦＣ８２にはＴＥＳＴ

コマンドのコマンドコード＄３Ｆが、そして、＄ＦＣ８３～＄ＦＣ８Ａにはパスワード（ＦＭ—7では何でも使えますので、こうしてみました）が格納されます。そして＄ＦＣ８Ｂからにサブコマンドが格納されます。ここでは＄Ｄ３８Ｆからのマシン語サブルーチンを実行するようになっています。図でもわかるように、マシン語プログラムはメイン側で＄ＦＣ８Ｆから格納されています。ＪＳＲサブコマンドでは、アドレスをサブＣＰＵ側のアドレスで指定するので＄Ｄ３８Ｆとしなければいけません。次の60行めからがサブＣＰＵによって実行されるマシン語サブルーチンです。

60行めでは、キーボード右上のＩＮＳキー横の発光ダイオードをつけています。これは、サブＣＰＵがこのマシン語サブルーチンを実行している

メイン		サブ	
＄ＦＣ８０	＄００	＄Ｄ３８０	
1	＄００	1	
2	＄３Ｆ	2	コマンドコード
3	'Ｆ	3	パスワード
4	'Ｍ	4	
5	'７	5	
6	'ハ	6	
7	'ス	7	
8	'コ	8	
9	'゛	9	
A	'イ	A	
B	＄９３	B	サブコマンド
C	＄Ｄ３	C	
D	＄８Ｆ	D	
E	＄９０	E	
F	＄Ｂ６	F	ＬＤＡ
＄ＦＣ９０	＄Ｄ４	＄Ｄ３９０	＄Ｄ４０Ｄ
1	＄０Ｄ	1	
2	＄Ｃ６	2	ＬＤＢ＃
	⋮		
＄ＦＣＡ８	＄０Ｄ	＄Ｄ３Ａ８	
9	＄３９	9	ＲＴＳ

図Ｅ－８　メインとサブの対応表

間は、発光ダイオードをつけておくことによって今サブＣＰＵが何をしているのかを視覚的にみえるようにしたものです。＄Ｄ４０Ｄは、サブＣＰＵのＩ／Ｏのアドレスで、このアドレスを読み出すと発光ダイオードが点灯します。

61行目からは、62行目で＄Ｄ４０３をリードすることにより一定時間ブザーを鳴らします（Ｉ／Ｏアドレスの詳細は、フェーズⅢを参照してください)。そして、時間待ちをするということを５回繰り返します。

そして、70行目で、＄Ｄ４０Ｄに書き込む（値はとにかく書き込めば何でもけっこうです）ことによって発光ダイオードを消し、71行目でサブルーチンを終了します。

こうすることによって、サブＣＰＵに独自のプログラムを実行させることができるわけです。このプログラムはぜひ実行させてください。

さて、ここで１つだけこのようなサブＣＰＵによって実行されるプログラムを作成する際の注意点を述べておきます。図Ｅ—7のプログラムのうちサブＣＰＵによって実行される部分は＄５０５１番地以降でした。しかし実際にサブＣＰＵで実行されるときの番地は、＄Ｄ３８Ｆ番地以降です。この様にプログラムを作成する時のアドレスと実行される時のアドレスとは異っています。ですから**サブＣＰＵで実行される部分のプログラムは、必ずポジションインディペンデント（位置独立）に作らなければなりません。**

4. メインＣＰＵ側プログラム

それでは、前章のゲームプログラムの高速化を考えていきましょう。まず、表示するデータと専用のプログラムのおき場所を考えなくてはなりません。図Ｅ—3のメモリマップをみるとわかると思いますが、サブＣＰＵのメモリは全て既に使われていて、余っているところはありません。だからといってデータやプログラムはいつも共有ＲＡＭ上というわけにはいきません（なんといっても共有ＲＡＭは128バイトしかありませんから)。

実は、サブＣＰＵ上のメモリのうち、制限を課せば、自由に使えるところがあります。それはコンソールバッファで、文字の表示さえしなければ、この部分は使用されません。ですから文字の表示をゲームの最中に行わないようにすれば、この部分にプログラムやデータを置くことができます。

それでは、まずプログラムやデータをこのコンソールバッファ（＄Ｃ０００～）に転送するプログラムを作ってみましょう。サブＣＰＵ上でのデータやプログラムの転送には、ＭＯＶＥサブコマンドを用います。つまり、共有ＲＡＭから＄Ｃ０００からへ順々に転送していけばよいわけです。

そこでここでは、転送するデータを64バイトごとに区切って順々に転送することにします。図Ｅ―9がそのプログラムです。

①でまずカーソルを消しています。これについては既に述べました。

②で、Ｙレジスタにサブ ＣＰＵ側の転送先のアドレスを代入し、③でＸレジスタにメインＣＰＵ側の転送元のアドレスを代入します。そして④で転送するブロック数（64バイトが１ブロック）をＢレジスタにセットしています。

⑤が、ブロック数だけ、ＴＯＳＵＢをコールして転送を行っている場所です。そして、⑥では転送したプログラムを作って画面に試表示している部分です。

⑦は、カーソル消去、⑧は試し表示用のデータです。

ＴＯＳＵＢからが１ブロック（64バイト）を転送するルーチンです。

⑨でサブＣＰＵを停止します。⑩では、ＴＥＳＴコマンドを設定するためのデータのアドレスをセットします。⑪では、そのデータを共有ＲＡＭにセットします。ここでＵレジスタは、共有ＲＡＭで次にデータをセットするところを示しています。

〔図Ｅ－９　転送プログラム〕

```
*
MAIN      LEAX  CUROFF,PCR
        ① BSR   TRANS
        ② LDY   #$C000
        ③ LEAX  SUBPRO,PCR
        ④ LDB   #(SUBPRE-SUBPRO)/64+1
          PSHS  B
MAIN1     BSR   TOSUB
        ⑤ DEC   ,S
          BNE   MAIN1
          PULS  B
          LEAX  SUBC,PCR
        ⑥ BSR   TRANS
          RTS

CUROFF    FCB   COFFE-COFFS
COFFS   ⑦ FCB   0,0,$0C,%011110
COFFE     EQU   *

SUBC      FCB   SUBCE-SUBCS
SUBCS     FCB   0,0,$3F
          FCC   /NAKAMURA/
        ⑧
          FCB   $93,$C0,$00,$90
          FCB   0,0,40,15
SUBCE     EQU   *

TOSUB     PSHS  X
        ⑨ BSR   STOP
        ⑩ LEAX  TESTCS,PCR
          LDU   #SRAM
          LDB   #TESTCE-TESTCS
TOSUB1    LDA   ,X+
        ⑪ STA   ,U+
          DECB
          BNE   TOSUB1
        ⑫ STY   ,U++
          LDD   #64
        ⑬ STD   ,U++
          LDA   #$90
        ⑭ STA   ,U+
          PULS  X
TOSUB2    LDA   ,X+
        ⑮ STA   ,U+
          LEAY  1,Y
          DECB
          BNE   TOSUB2
        ⑯ BRA   RUN

TESTCS    FCB   0,0,$3F
          FCC   /SUBCPUPR/
        ⑰
          FCB   $91
          FDB   $D393
TESTCE    EQU   *
```

図Ｅ－10　転送プログラムの概念図

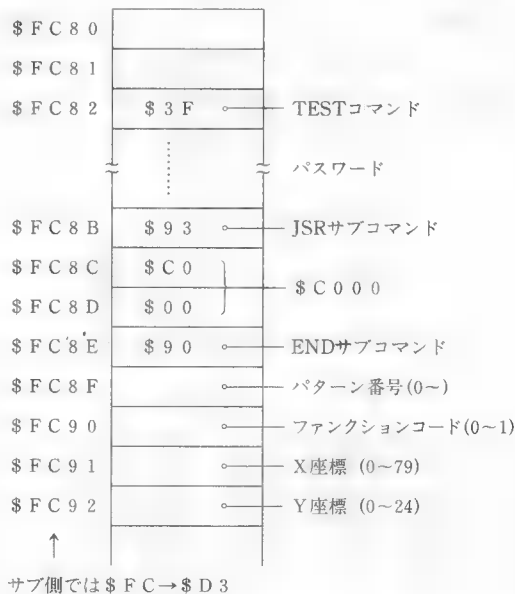
⑫では、サブＣＰＵ側での転送先のアドレス(ＭＯＶＥコマンドの２つめのパタメータ）をセットしています。そして、⑬で転送バイト数（64バイト）をセットし、⑭で、ＥＮＤサブコマンドをセットします。

⑮では、Ｘレジスタの示すところからＵレジスタの示す共有ＲＡＭ上へ64バイトを転送する部分です。同時に、Ｙレジスタもインクリメントしています。

最後に⑯でサブＣＰＵの停止を解除して終了します。

⑰は、ＴＥＳＴコマンドのためのデータです。このデータでは、ＭＯＶＥサブコマンドのコード＄９１と、転送元（サブＣＰＵ側）のアドレスまでをセットします。

5. サブＣＰＵ側プログラム

次にサブＣＰＵ側のプログラムの製作に入るわけですが、まず、どういったルーチンにするかを決めなくてはなりません。そこで、サブＣＰＵは与えられたパターン番号と座標からサブＣＰＵ側

図Ｅ－11　サブCPUプログラムの設計

〔図Ｅ－12　サブＣＰＵプログラム〕

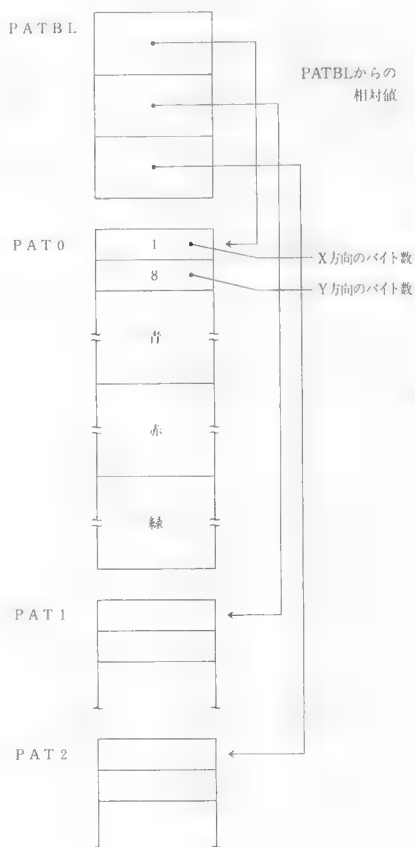
```
******************
*
* PROGRAM EXECUTE
*       BY
*    SUB CPU
*
******************
VRAM     EQU     $D409
INSLED   EQU     $D40D
*
PATRN    EQU     $D38F
FUNC     EQU     PATRN+1
POSX     EQU     FUNC+1
POSY     EQU     POSX+1
*
SUBPRO   EQU     *
*
      ① LDA      VRAM
      ② LDA      INSLED

       ┌LDB      PATRN
       │ASLB
      ③│LEAY     PATBL,PCR
       │LDD      B,Y
       └LEAY     D,Y

       ┌LDA      POSY
       │LDB      #8
      ④│MUL
       │LDA      #80
       │MUL
       └TFR      D,X
      ⑤┌LDB     POSX
       └ABX

      ⑥┌LDD     ,Y++
       └STD      XLEN,PCR

      ⑦┌LDB     #3
       └PSHS     B
PUT1   ┌PSHS     X
      ⑧│LDB      YLEN,PCR
       └PSHS     B
PUT2   ┌PSHS     X
       │LDB      XLEN,PCR
PUT3   │LDA      ,Y+
       │TST      FUNC
      ⑨│BEQ      PUT4
       │CLRA
PUT4   │STA      ,X+
       │DECB
       │BNE      PUT3
       └PULS     X
       ┌LEAX     80,X
      ⑩│DEC      ,S
       │BNE      PUT2
       └PULS     B
       ┌PULS     X
       │LEAX     $4000,X
      ⑪│DEC      ,S
       │BNE      PUT1
       └PULS     B

       ┌STA      INSLED
      ⑫│STA      VRAM
       └RTS

XLEN  ⑬┌RMB     1
YLEN   └RMB      1
```

に転送されているパターンを表示するプログラムにすることにします。

そして、パターン番号などのデータは、図E―11のように与えることにします。つまり、専用のプログラムを＄Ｃ０００からに転送しておき、サブＣＰＵへの指令は形式上ＴＥＳＴコマンドの形を取り、サブコマンド列の後に表示に関するパラメータを並べるという方法を取ります。

パターン番号は、サブＣＰＵに転送した表示データに０から番号をつけて、その番号により表示するパターンを選択します。

ファンクションコードは、そのパターンを表示するのか消去するのかを指定するパラメータです。ＰＧＢＬＫ１などにあわせて、０で表示、１で消去にします。

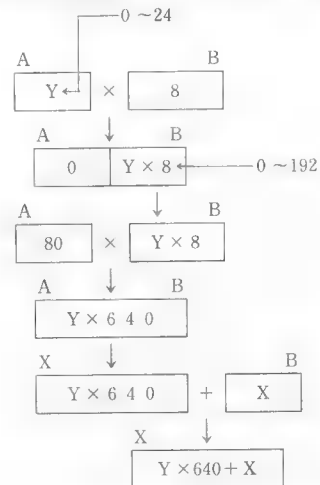
Ｘ、Ｙ座標は、パターンを表示する位置を指定するパラメータです。ここでは、プログラムを組みやすくするために、各座標はキャラクタ座標で指定することにします。

ですから＄Ｃ０００からのサブ側のプログラムは＄Ｄ３８Ｆ～＄Ｄ３９２のパラメータを読み取って指定されたパターンを画面に表示または消去するプログラムとなります。それでは、そのプログラム（図Ｅ―12）を解説していきます。

①では、ＶＲＡＭアクセスフラグというものをセットしています。このＶＲＡＭアクセスフラグというのはＶＲＡＭに値を書き込んだり、読み出したりするときには事前にセットしておかなければいけないフラグです。これはハードウェア的な面からくる要請です。詳しくはフェーズIIIを参照してください。

②では、前と同じくＩＮＳキー横の発光ダイオードを点灯させています。

③では、与えられたパターン番号から各パターンの先頭アドレスを求めています。この部分のデータ構造を示したものが図Ｅ―13です。この方法は、前章の移動ルーチンのところで用いたものと同じです。

④では、与えられたＹ座標から、表示位置のある行の最右のアドレスを求めています。方法は、Ｙ座標を640倍してＸレジスタにセットしています。まず８倍してから80倍していますがこうしなければいけない理由は各自考えてください。そし

図E－13　パターンデータの構造

図E－14　アドレスの算出

て⑤でX座標を加算して表示位置のアドレスを得ます（図E―14参照）。

⑥では、表示するデータの大きさをＸＬＥＮ、ＹＬＥＮにセットしています。この大きさは、X、

〔図E－15　実験プログラム〕

```
  1                         *
  2                         * EQUATES
  3                         *
  4                  FD05   HALT     EQU      $FD05
  5                  FC80   SRAM     EQU      $FC80
  6                         *
  7   5000                           ORG      $5000
  8   5000 20    2B         START    BRA      MAIN
  9                         *
 10                         * SUBCPU CONTROL PROGRAMS
 11                         *
 12                         * STOP SUB CPU
 13   5002 34    02         STOP     PSHS     A
 14   5004 B6    FD05       STOP1    LDA      HALT
 15   5007 2B    FB                  BMI      STOP1
 16   5009 86    80                  LDA      #$80
 17   500B B7    FD05                STA      HALT
 18   500E B6    FD05       STOP2    LDA      HALT
 19   5011 2A    FB                  BPL      STOP2
 20   5013 35    82                  PULS     A,PC
 21                         * RUN SUB CPU
 22   5015 7F    FD05       RUN      CLR      HALT
 23   5018 39                        RTS
 24                         * TRANS
 25   5019 34    56         TRANS    PSHS     D,X,U
 26   501B 8D    E5                  BSR      STOP
 27   501D E6    80                  LDB      ,X+
 28   501F CE    FC80                LDU      #SRAM
 29   5022 A6    80         TRANS1   LDA      ,X+
 30   5024 A7    C0                  STA      ,U+
 31   5026 5A                        DECB
 32   5027 26    F9                  BNE      TRANS1
 33   5029 8D    EA                  BSR      RUN
 34   502B 35    D6                  PULS     D,X,U,PC
```

この間に
　図E－9転送プログラム
　図E－12サブCPUプログラム
　図E－17の中の412行から538行まで
が順に入ります

```
284
285                521C   SUBPRE   EQU      *
286
287                                END      START

0 ERROR(S) DETECTED

SYMBOL TABLE:

COFFE  5053   COFFS  504F   CUROFF 504E   FUNC   D390   HALT   FD05
INSLED D40D   MAIN   502D   MAIN1  503F   PAT0   510E   PAT1   5128
PAT2   51BA   PATBL  5108   PATRN  D38F   POSX   D391   POSY   D392
PUT1   50CD   PUT2   50D5   PUT3   50DB   PUT4   50E3   RUN    5015
SRAM   FC80   START  5000   STOP   5002   STOP1  5004   STOP2  500E
SUBC   5053   SUBCE  5067   SUBCS  5054   SUBPRE 521C   SUBPRO 50A2
TESTCE 50A2   TESTCS 5094   TOSUB  5067   TOSUB1 5074   TOSUB2 5089
TRANS  5019   TRANS1 5022   VRAM   D409   XLEN   5106   YLEN   5107
```

Y方向のバイト数で示されています。Y方向は1ドット＝1バイトですが、X方向は、8ドット＝1バイトですので､実際に表示するドット数は(X方向バイト数×8)×（Y方向バイト数）となります。各パターンのデータは最初の2バイトにこのバイト数があり､その後に､XLEN×YLEN×3バイトのデータが続きます。データは青の画面用のデータ、赤、緑の順に並べておきます。

⑦と⑪で3画面分のループを構成しています。各画面ごとに表示するアドレスを$4000ずつずらしています。

⑧と⑩では、Y方向の表示ループを構成しています。1行ごとにアドレスは80だけ増加します。

⑨が、X方向の表示を行う部分です。ここでは、Yレジスタが示すパターンデータをXレジスタが示す画面のアドレスへ転送しています。この際、ファンクションコードをみて、これが1ならば、画面へは0を書き込みます。

そして、最後に発光ダイオードを消し、VRAMアクセスフラグをリセットして終了します(⑫)。

⑬は、X、Y各方向のバイト数を退避しておくところです。

既に述べましたが、このプログラムはポジションインディペンデントになっていることに注意してください。

図E―15に、この部分（サブCPUに転送してプログラムを実行させる）を実験するためのプログラムを示します。ソースリストは重複となるので一部を省略しました。そのかわり、図E―16にこのプログラムのダンプリストをあげておきます。

このプログラムは、サブCPUにプログラムを転送したあと、そのパターンの一つを画面中央付近に表示します。$5063番地を書き換えると表示するパターン番号がかわります。また$5065、$5066番地は、表示位置です。いろいろなパターンを登録して表示させてみるのもよい

〔図E－16　実験プログラムダンプリスト〕

```
5000: 20 2B 34 02 B6 FD 05 2B FB 86 80 B7 FD 05 B6 FD :D1   +4.カ人.+町■_キ人.カ人
5010: 05 2A FB 35 82 7F FD 05 39 34 56 8D E5 E6 80 CE :CB  .*町5_.人.94V■■■_ホ
5020: FC 80 A6 80 A7 C0 5A 26 F9 8D EA 35 D6 30 8D 00 :C1  ﾈ_ヲ_ｧﾀZ&ホ■◆530■.
5030: 1D 8D E6 10 8E C0 00 30 8D 00 67 C6 06 34 04 8D :A3  .■■.■ﾀ.0■.gニ.4.■
5040: 26 6A E4 26 FA 35 04 30 8D 00 08 8D CC 39 04 00 :28  &j■&E5.0■..■79..
5050: 00 0C 1E 13 00 00 3F 4E 41 4B 41 4D 55 52 41 93 :5F  ......?NAKAMURAト
5060: C0 00 90 00 00 28 0F 34 10 8D 97 30 8D 00 25 CE :9F  ﾀ.ｰ..(.4.■ 10■.%ホ
5070: FC 80 C6 0E A6 80 A7 C0 5A 26 F9 10 AF C1 CC 00 :A2  ﾈ_ニ.ヲ_ｧﾀZ&ホ.ｯﾁﾌ.
5080: 40 ED C1 86 90 A7 C0 35 10 A6 80 A7 C0 31 21 5A :E9  @ﾛﾁ■ｰｧﾀ5.ヲ_ｧﾀ1!Z
5090: 26 F7 20 81 00 00 3F 53 55 42 43 50 55 50 52 91 :02  &■ _..?SUBCPUPRｰ
50A0: D3 93 B6 D4 09 B6 D4 0D F6 D3 8F 58 31 8D 00 58 :56  モトカヤ.カヤ.分モ+X1■.X
50B0: EC A5 31 AB B6 D3 92 C6 08 3D 86 50 3D 1F 01 F6 :BC  ●・1ｵカモｰニ.=■P=..分
50C0: D3 91 3A EC A1 ED 8D 00 3D C6 03 34 04 34 10 E6 :0D  モｰ:●｡ロ■.=ニ.4.4.■
50D0: 8D 00 34 34 04 34 10 E6 8D 00 2B A6 A0 7D D3 90 :01  ■.44.4.■■.+ヲ }モｰ
50E0: 27 01 4F A7 80 5A 26 F3 35 10 30 88 50 6A E4 26 :D2  '.Oｧ_Z&■5.0| Pj■&
50F0: E4 35 04 35 10 30 89 40 00 6A E4 26 D0 35 04 B7 :8F  ■5.5.0| @.j■&ミ5.キ
5100: D4 0D B7 D4 09 39 FF FF 00 06 00 20 00 B2 01 08 :8D  ヤ.キヤ.9  ... .イ..
5110: 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 :80  ................
5120: 18 18 18 18 18 18 18 18 03 10 00 18 00 00 18 00 :03  ................
5130: 00 18 00 00 24 00 00 7E 00 18 66 18 3C 66 3C 7E :AC  ....$..~..f.<f<~
5140: 66 7E FF E7 FF FF E7 FF FF FF FF FF 00 FF E7 00 :90  f~ ■  ■       . ■.
5150: E7 C3 00 C3 81 00 81 81 00 81 00 18 00 00 18 00 :A1  ■テ.テ_.__._......
5160: 00 18 00 00 3C 00 00 7E 00 18 66 18 24 42 24 42 :34  ....<..~..f.$B$B
5170: 42 42 81 81 81 81 81 81 81 FF 81 99 00 99 BD 00 :7A  BB_______ _ｺ.ｺｽ.
5180: BD FF 00 FF BD 00 BD 99 00 99 00 18 00 00 18 00 :97  ｽ . ｽ.ｽｺ.ｺ......
5190: 00 18 00 00 3C 00 00 7E 00 18 7E 18 24 7E 24 66 :AC  ....<..~..~.$~$f
51A0: 7E 66 A5 FF A5 A5 FF A5 A5 FF A5 BD 00 BD A5 00 :DE  ~f・ ・・ ・・ ・ｽ.ｽ・.
51B0: A5 C3 00 C3 81 00 81 81 00 81 02 10 FF FF 70 0E :BD  ・テ.テ_.__._..  p.
51C0: 38 1C 1C 38 0E 70 07 E0 03 C0 00 0C 00 0C 03 C0 :AB  8..8.p.ニ.ﾀ.....ﾀ
51D0: 07 E0 0E 70 1C 38 38 1C 70 0E FF FF 00 00 70 0E :07  .ニ.p.88.p.  ..p.
51E0: 38 1C 1C 38 0E 70 07 E0 03 C0 30 00 30 00 03 C0 :F3  8..8.p.ニ.ﾀ0.0..ﾀ
51F0: 07 E0 0E 70 1C 38 38 1C 70 0E 00 00 00 00 00 00 :8B  .ニ.p.88.p......
5200: 00 00 00 00 00 00 00 00 00 00 0F FC 0F FC 00 00 :16  ...........ﾈ.ﾈ..
5210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :00  ................
```

でしょう（$5108からが、PATBLになっています)。

6. ゲームプログラムの変更

これで高速化の準備は整いました。あとは、これらのルーチンを前章のプログラムの中に組み込めばよいわけです。

図E—17が、これらのルーチンを組み込んだプログラムの全リストです。根本的な考え方は全く前章のものと同じですから、ここでは高速化ルーチンの組み込みによって変更した部分のみ解説します（解説はリスト最左の行番号を参照して行います)。

まず高速化されたので敵の数を4個から6個に増やしました（34行)。

メインルーチン（40〜104行）は大幅に変更しました。高速化されたことにより全体の動きが前章のままのメインルーチンでは多少ぎこちなくなってしまいました。そこで、各ルーチンはメインルーチンのループを回るごとに毎回実行されるのではなく、何回かおきに実行させるようにしました。このためのカウンタがCOUNT0〜3で、各ルーチンはこのカウンタを（メインルーチンのループごとに）デクリメント（−1）して、0になったときのみ実行されるようにしました。各ルーチンの実行頻度は、敵の移動が6回に1回、砲台の移動が3回に1回、光線の移動が2回に1回、星の移動が10回に1回となっています。この実行頻度を変えれば、光線の速度などを変えることもできます。

249行では、INIT（初期化）ルーチンの最後でSUBSUB(図E—9のプログラム、ラベルがかわっている）を呼び出して、サブCPUへ専用のプログラムを転送しています。

295〜538行が、今回作成した高速化のためのルーチンです。内容は、図E—9、図E—12のものとほぼ同じです。異なっているのは、386、394行です。図E—12のプログラムでは、ルーチンの最初と最後にあったものが、ルーチンの中程にきています。これは、VRAMアクセスフラグをセットすると、サブCPUの実行速度が半分以下に落ちるため、遅くなる部分を少しでも減らそうという処置です。

後は、各パターンの表示・消去ルーチンの変更です。560〜571行が砲台、621〜632行が光線、729〜740行が敵の表示、消去部分です。それぞれのルーチンは、前章のときに必要だったパターンの右下の座標の指定がはいらないのでその分簡単になっています。各ルーチンは、ファンクションコードとX、Y座標をセットしてサブCPUに表示をさせています。

この他にもメッセージなどで若干の変更がありますが解説する必要はないでしょう。図E—18にダンプリストをあげておきます。

〔図E—17 高速版ゲーム全リスト〕

```
  1      ******************************
  2      *
  3      * SAMPLE GAME PROGRAM
  4      *        FOR
  5      *    FM-7/NEW-7/77
  6      *    ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏﾆｭｱﾙ
  7      *
  8      *  presented by H.NAKAMURA
  9      *
 10      *    Z I G   Z A P '
 11      *      PART  II
 12      *     ( fast version )
 13      *       Ver 1.00
 14      *
 15      *  DATE 58/05/20 11:59
 16      *
 17      ******************************
```

```
18   2000                      ORG    $2000      program start from $2000
19   2000 20  02        START  BRA    GAME       branch to main program
20                      *
21                      * EQUATES
22                      *
23               FD05   HALT   EQU    $FD05      I/O addr. of subCPU control
24               FC80   SRAM   EQU    $FC80      top address of shared RAM
25               FD0D   PSGC   EQU    $FD0D      PSG control register
26               FD0E   PSGD   EQU    $FD0E      PSG data register
27               FD01   KEYBRD EQU    $FD01      keyboard address of mainCPU
28               FD04   BREAK  EQU    $FD04      break key check address
29
30               007B   RNDC   EQU    123        random number constant
31                      *
32                      * CONSTANT
33                      *
34   2002 06            ENEMYN FCB    6          number of enemies
35   2003 0A            MISILN FCB    10         number of missiles
36
37                      *
38                      * MAIN ROUTINE
39                      *
40               2004   GAME   EQU    *          game routine
41   2004 17  0158             LBSR   INIT       initialize
42   2007 30  8D 0083          LEAX   COUNT0,PCR counter reset
43   200B CC  0101             LDD    #$0101
44   200E ED  84               STD    ,X
45   2010 ED  02               STD    2,X
46
47   2012 6A  8D 0078   MLOOP  DEC    COUNT0,PCR
48   2016 26  12               BNE    ML1
49   2018 86  06               LDA    #6         1time / 6loop
50   201A A7  8D 0070          STA    COUNT0,PCR
51   201E 17  04C1             LBSR   ENEMY      enemy move
52   2021 6D  8D 005C          TST    SIGNAL,PCR test signal
53   2025 26  35               BNE    ML4
54   2027 17  0582             LBSR   CHECK      check enemy & missile
55   202A 6A  8D 0061   ML1    DEC    COUNT1,PCR
56   202E 26  09               BNE    ML2
57   2030 86  03               LDA    #3         1time / 3loop
58   2032 A7  8D 0059          STA    COUNT1,PCR
59   2036 17  03CE             LBSR   CANNON     cannon move
60   2039 6A  8D 0053   ML2    DEC    COUNT2,PCR
61   203D 26  0C               BNE    ML3
62   203F 86  02               LDA    #2         1time / 2loop
63   2041 A7  8D 004B          STA    COUNT2,PCR
64   2045 17  040C             LBSR   MSIL       missile move
65   2048 17  0561             LBSR   CHECK      check missile & enemy
66   204B 6A  8D 0042   ML3    DEC    COUNT3,PCR
67   204F 26  C1               BNE    MLOOP
68   2051 86  0A               LDA    #10        1time / 10loop
69   2053 A7  8D 003A          STA    COUNT3,PCR
70   2057 17  05C0             LBSR   STAR       star move
71   205A 20  B6               BRA    MLOOP
72   205C C6  06        ML4    LDB    #6         6th message 'GAME OVER ..'
73   205E 17  0096             LBSR   PRINT      score output
74   2061 17  00B6             LBSR   SCOUT      wait keyin
75  >2064 17  006F      GAMEX  LBSR   KEYIN
76   2067 81  59               CMPA   #'Y        if 'y' or 'Y' then replay
77   2069 27  99               BEQ    GAME
78   206B 81  79               CMPA   #'y
79   206D 27  95               BEQ    GAME
80   206F 81  4E               CMPA   #'N        if 'n' OR 'N' then end
81   2071 27  04               BEQ    ENDPRO
82   2073 81  6E               CMPA   #'n
83   2075 26  ED               BNE    GAMEX
84                      *
85   2077 30  8D 0154   ENDPRO LEAX   CLSCOM,PCR
```


```
71   205A 20  B6               BRA    MLOOP
72   205C C6  06        ML4    LDB    #6         6th message 'GAME OVER ..
```

```
 73   205E 17   0096            LBSR   PRINT      score output
 74   2061 17   00B6            LBSR   SCOUT      wait keyin
 75  >2064 17   006F    GAMEX   LBSR   KEYIN
 76   2067 81   59              CMPA   #'Y        if 'y' or 'Y' then replay
 77   2069 27   99              BEQ    GAME
 78   206B 81   79              CMPA   #'y
 79   206D 27   95              BEQ    GAME
 80   206F 81   4E              CMPA   #'N        if 'n' OR 'N' then end
 81   2071 27   04              BEQ    ENDPRO
 82   2073 81   6E              CMPA   #'n
 83   2075 26   ED              BNE    GAMEX
 84                     *
 85   2077 30   8D 0154 ENDPRO  LEAX   CLSCOM,PCR
 86  >207B 17   002B            LBSR   TRANS
 87   207E 1C   AF              ANDCC  #$AF       interrupt enable
 88   2080 39                   RTS               end of this game
 89
 90                     *
 91                     * MAIN ROUTINE WORK
 92                     *
 93   2081              SIGNAL  RMB    1          gameover then not 0
 94   2082              SCORE   RMB    2          score (1 enemy 10 point)
 95   2084              CANX    RMB    1          locate X of cannon
 96   2085              MISBUF  RMB    2          missile buffer top addr.
 97   2087              EMYBUF  RMB    2          enemies buffer top addr.
 98   2089 0001 0000    RNDWRK  FDB    1,0        random number works
 99   208D              KEYFLG  RMB    1          key flag for break key
100
101   208E              COUNT0  RMB    1
102   208F              COUNT1  RMB    1
103   2090              COUNT2  RMB    1
104   2091              COUNT3  RMB    1
105                     *
106                     * SUBCPU CONTROL PROGRAMS
107                     *
108                     * STOP SUB CPU
109   2092 34   02      STOP    PSHS   A          save Areg
110   2094 B6   FD05    STOP1   LDA    HALT       ready check
111   2097 2B   FB              BMI    STOP1      busy then wait
112   2099 86   80              LDA    #$80       halt (stop) subCPU
113   209B B7   FD05            STA    HALT
114   209E B6   FD05    STOP2   LDA    HALT       halt check
115   20A1 2A   FB              BPL    STOP2      not busy then wait
116   20A3 35   82              PULS   A,PC       return
117                     * RUN SUB CPU
118   20A5 7F   FD05    RUN     CLR    HALT       run subCPU (bit7 clear)
119   20A8 39                   RTS               return
120                     * TRANS
121   20A9 34   56      TRANS   PSHS   D,X,U      save registers
122   20AB 8D   E5              BSR    STOP       stop subCPU
123   20AD E6   80              LDB    ,X+        bytes of transfer data
124   20AF CE   FC80            LDU    #SRAM      top address of shared RAM
125   20B2 A6   80      TRANS1  LDA    ,X+        transfer
126   20B4 A7   C0              STA    ,U+
127   20B6 5A                   DECB              done ?
128   20B7 26   F9              BNE    TRANS1     no, then loop
129   20B9 8D   EA              BSR    RUN        run subCPU
130   20BB 35   D6              PULS   D,X,U,PC   return
131
132                     *
133                     * PSG CONTROL ROUTINE
134                     *
135   20BD 34   06      PSG     PSHS   D          save register
136   20BF B7   FD0E            STA    PSGD       PSG register no. set
137   20C2 86   03              LDA    #$03       register set command
138   20C4 B7   FD0D            STA    PSGC       command set
139   20C7 7F   FD0D            CLR    PSGC
140   20CA F7   FD0E            STB    PSGD       command clear
141   20CD 4A                   DECA              data set command ($02)
142   20CE B7   FD0D            STA    PSGC       command set
```

```
143   20D1 7F    FD0D               CLR    PSGC       command clear
144   20D4 35    86                 PULS   D,PC       return
145
146                         *
147                         * KEYIN ROUTINE
148                         *
149   20D6 34    14         KEYIN   PSHS   B,X        save registers
150   20D8 30    8D 0016            LEAX   INKEYC,PCR inkey command addr.
151  >20DC 17    FFCA               LBSR   TRANS      transfer to subCPU
152  >20DF 17    FFB0               LBSR   STOP       stop subCPU for data read
153   20E2 B6    FC83               LDA    SRAM+3     get key data
154   20E5 F6    FC80               LDB    SRAM       request flag on
155   20E8 CA    80                 ORB    #%10000000
156   20EA F7    FC80               STB    SRAM
157  >20ED 17    FFB5               LBSR   RUN        restart subCPU
158   20F0 35    94                 PULS   B,X,PC     return
159   20F2 04               INKEYC  FCB    EINKYC-SINKYC
160   20F3 00 00 29 03      SINKYC  FCB    0,0,$29,%00000011 INKEY with wait
161                  20F7   EINKYC  EQU    *
162
163                         *
164                         * PRINT OUT MESSAGE
165                         *
166   20F7 34    56         PRINT   PSHS   D,X,U      save registers
167  >20F9 17    FF96               LBSR   STOP       stop subCPU
168   20FC CE    FC82               LDU    #SRAM+2    load destination
169   20FF 86    03                 LDA    #$03       PUTC request
170   2101 A7    C0                 STA    ,U+
171   2103 58                       ASLB              string no *2
172   2104 30    8D 0648            LEAX   MSGTBL,PCR table addr.
173   2108 EC    85                 LDD    B,X        load offset
174   210A 30    8B                 LEAX   D,X        load message addr.
175   210C E6    84                 LDB    ,X         load string length
176   210E A6    80         PRINT1  LDA    ,X+        transfer string
177   2110 A7    C0                 STA    ,U+
178   2112 5A                       DECB
179   2113 2A    F9                 BPL    PRINT1
180  >2115 17    FF8D               LBSR   RUN        restart subCPU
181   2118 35    D6                 PULS   D,X,U,PC   return
182
183                         *
184                         * SCORE OUTPUT
185                         *
186   211A 33    8D 002D    SCOUT   LEAU   NUMTB,PCR load convert table
187   211E 30    8D 0038            LEAX   NUMBUF+5,PCR load buffer addr.
188   2122 86    05                 LDA    #5         length 5
189   2124 34    02                 PSHS   A
190   2126 86    2F         SCOUT1  LDA    #'0-1      set chr.
191   2128 34    02                 PSHS   A
192   212A EC    8D FF54            LDD    SCORE,PCR load score
193   212E 6C    E4         SCOUT2  INC    ,S         inc chr.
194   2130 A3    C4                 SUBD   ,U         score - 10^x
195   2132 24    FA                 BCC    SCOUT2
196   2134 E3    C1                 ADDD   ,U++
197   2136 ED    8D FF48            STD    SCORE,PCR set mod.
198   213A 35    02                 PULS   A          pull chr.
199   213C A7    80                 STA    ,X+        set character
200   213E 6A    E4                 DEC    ,S         dec counter
201   2140 26    E4                 BNE    SCOUT1
202   2142 35    02                 PULS   A
203   2144 30    8D 000D            LEAX   NUMBUF,PCR print out number
204   2148 16    FF5E               LBRA   TRANS
205   214B 2710 03E8        NUMTB   FDB    10000,1000,100,10,1 convert table
206   2155 09               NUMBUF  FCB    9
207   2156 00 00 03 05              FCB    0,0,$03,5
208   215A 30 30 30 30              FCC    /00000/    character buffer
209
210                         *
211                         * INITIALIZE ROUTINE
212                         *
```

```
213    215F 1A    50          INIT    ORCC   #$50        mask FIRQ/IRQ
214    2161 30    8D 005E             LEAX   SCRNI,PCR screen initialize
215    2165 17    FF41                LBSR   TRANS
216    2168 17    0081                LBSR   INTMSG      output opening mes.
217    216B 17    FF68        INIT1   LBSR   KEYIN       wait keyin
218    216E 81    20                  CMPA   #'          SPACE
219    2170 26    F9                  BNE    INIT1       if not space then loop
220    2172 6F    8D FF0B             CLR    SIGNAL,PCR clear signal
221    2176 CC    0000                LDD    #0          clear score
222    2179 ED    8D FF05             STD    SCORE,PCR
223    217D 30    8D 07EF             LEAX   BUFFER,PCR set buffer addr.
224    2181 AF    8D FF00             STX    MISBUF,PCR missile buffer
225    2185 E6    8D FE7A             LDB    MISILN,PCR reserve misbuf
226    2189 6F    80          INIT2   CLR    ,X+         3byte/1missile
227    218B 6F    80                  CLR    ,X+
228    218D 6F    80                  CLR    ,X+
229    218F 5A                        DECB
230    2190 26    F7                  BNE    INIT2
231    2192 AF    8D FEF1             STX    EMYBUF,PCR set emybuf
232    2196 E6    8D FE68             LDB    ENEMYN,PCR initialize emybuf
233    219A 6F    84          INIT3   CLR    ,X          clear X,Y
234    219C 6F    01                  CLR    1,X
235    219E AF    02                  STX    2,X         point to 0
236    21A0 30    04                  LEAX   4,X         next enemy
237    21A2 5A                        DECB
238    21A3 26    F5                  BNE    INIT3
239    21A5 86    28                  LDA    #40         set cannon X
240    21A7 A7    8D FED9             STA    CANX,PCR
241    21AB 30    8D 0027             LEAX   PSGDAT,PCR PSG initialize
242    21AF C6    0B                  LDB    #11         number of PSG data
243    21B1 EC    81          INIT4   LDD    ,X++        get PSG data
244    21B3 17    FF07                LBSR   PSG
245    21B6 5A                        DECB
246    21B7 26    F8                  BNE    INIT4
247    21B9 30    8D 0012             LEAX   CLSCOM,PCR clear screen
248    21BD 17    FEE9                LBSR   TRANS
249   >21C0 16    006D                LBRA   SUBSUB      transfer to subCPU
250
251    21C3 0B                SCRNI   FCB    11          width 80,25 etc.
252    21C4 00 00 01 00               FCB    0,0,1,0,80,25,0,25,0,1,0
253    21CF 06                CLSCOM  FCB    6           clear screen
254    21D0 00 00 0D 00               FCB    0,0,$0D,0,0,7
255    21D6 07F9 0200         PSGDAT  FDB    $07F9,$0200,$0302,$0400
256    21DE 0504 0600                 FDB    $0504,$0600,$0810,$0910
257    21E6 0A10 0B00                 FDB    $0A10,$0B00,$0C10
258
259                           *
260                           * OPENING MESSAGES OUTPUT
261                           *
262                     21EC  INTMSG  EQU    *
263    21EC C6    03                  LDB    #3          for i=3 downto 1
264    21EE 30    8D 0029             LEAX   SYMCOM,PCR
265    21F2 34    04          INTMO   PSHS   B
266    21F4 E7    0C                  STB    12,X        symbol(160+i,0+i),..
267    21F6 CB    A0                  ADDB   #160
268    21F8 E7    0A                  STB    10,X
269    21FA 17    FEAC                LBSR   TRANS
270    21FD 35    04                  PULS   B
271    21FF 5A                        DECB
272    2200 26    F0                  BNE    INTMO
273    2202 6A    0C                  DEC    12,X        symbol(160,0),.. color=5
274    2204 6A    0A                  DEC    10,X
275    2206 86    05                  LDA    #5
276    2208 A7    04                  STA    4,X
277    220A 17    FE9C                LBSR   TRANS
278    220D 86    01                  LDA    #1          reset color=1
279    220F A7    04                  STA    4,X
280
281    2211 5F                        CLRB               print out mes0..mes5
282    2212 17    FEE2        INTM1   LBSR   PRINT
```

```
283    2215 5C                      INCB
284    2216 C1    06                CMPB    #6
285    2218 26    F8                BNE     INTM1
286    221A 39                      RTS
287
288    221B 14              SYMCOM  FCB     SCOM2-SCOM1
289    221C 00 00 19 01     SCOM1   FCB     0,0,$19,1
290    2220 00 00 05 03             FCB     0,0,5,3
291    2224 00A0 0000               FDB     160,0
292    2228 07                      FCB     7
293    2229 5A 49 47 20             FCC     /ZIG ZAP/
294                   2230  SCOM2   EQU     *
295                         *********************
296                         *
297                         * TRANSFER PROGRAM TO SUBCPU
298                         *
299    2230 30    8D 0017   SUBSUB  LEAX    CUROFF,PCR cursor off
300    2234 17    FE72              LBSR    TRANS
301    2237 108E C000               LDY     #$C000     subcpu prog. $C000-
302    223B 30    8D 004E           LEAX    SUBPRO,PCR
303    223F C6    06                LDB     #(SUBPRE-SUBPRO)/64+1 block no.
304    2241 34    04                PSHS    B
305    2243 8D    0B        SUBSB1  BSR     TOSUB      trans. 1 block
306    2245 6A    E4                DEC     ,S
307    2247 26    FA                BNE     SUBSB1
308    2249 35    84                PULS    B,PC
309
310    224B 04              CUROFF  FCB     COFFE-COFFS
311    224C 00 00 0C 1E     COFFS   FCB     0,0,$0C,%011110
312                   2250  COFFE   EQU     *
313
314    2250 34    10        TOSUB   PSHS    X          trans 1 block sub.
315    2252 17    FE3D              LBSR    STOP
316    2255 30    8D 0026           LEAX    TESTCS,PCR
317    2259 CE    FC80              LDU     #SRAM      transfer to SRAM
318    225C C6    0E                LDB     #TESTCE-TESTCS
319    225E A6    80        TOSUB1  LDA     ,X+
320    2260 A7    C0                STA     ,U+
321    2262 5A                      DECB
322    2263 26    F9                BNE     TOSUB1
323    2265 10AF C1                 STY     ,U++       subcpu addr.
324    2268 CC    0040              LDD     #64        length
325    226B ED    C1                STD     ,U++
326    226D 86    90                LDA     #$90       end of test-subcommand
327    226F A7    C0                STA     ,U+
328    2271 35    10                PULS    X
329    2273 A6    80        TOSUB2  LDA     ,X+        trans 1 block
330    2275 A7    C0                STA     ,U+
331    2277 31    21                LEAY    1,Y
332    2279 5A                      DECB
333    227A 26    F7                BNE     TOSUB2
334    227C 16    FE26              LBRA    RUN
335
336    227F 00 00 3F        TESTCS  FCB     0,0,$3F
337    2282 53 55 42 43             FCC     /SUBCPUPR/
338    228A 91                      FCB     $91        test-subcommand MOVE
339    228B D393                    FDB     $D393
340                   228D  TESTCE  EQU     *
341
342                         ******************
343                         *
344                         * PROGRAM EXECUTE
345                         *       BY
346                         *    SUB CPU
347                         *
348                         ******************
349                   D409  VRAM    EQU     $D409      VRAM flag
350                   D40D  INSLED  EQU     $D40D      INS LED
351                         *
352                   D38F  PATRN   EQU     $D38F      pattern no.
```

```
353                  D390  FUNC    EQU   PATRN+1    function code 0or1
354                  D391  POSX    EQU   FUNC+1     position X & Y
355                  D392  POSY    EQU   POSX+1
356                        *
357                  228D  SUBPRO  EQU   *
358                        *
359  228D B6  D40D                 LDA   INSLED
360
361  2290 F6  D38F                 LDB   PATRN      set pattern addr.
362  2293 58                       ASLB             in Yreg
363  2294 31  8D 005B              LEAY  PATBL,PCR
364  2298 EC  A5                   LDD   B,Y
365  229A 31  AB                   LEAY  D,Y
366
367  229C B6  D392                 LDA   POSY       set VRAM addr.
368  229F C6  08                   LDB   #8         in Xreg
369  22A1 3D                       MUL
370  22A2 86  50                   LDA   #80
371  22A4 3D                       MUL
372  22A5 1F  01                   TFR   D,X
373  22A7 F6  D391                 LDB   POSX
374  22AA 3A                       ABX
375
376  22AB EC  A1                   LDD   ,Y++       pattern length
377  22AD ED  8D 0040              STD   XLEN,PCR
378
379  22B1 C6  03                   LDB   #3         3 plane
380  22B3 34  04                   PSHS  B
381  22B5 34  10           PUT1    PSHS  X
382  22B7 E6  8D 0037              LDB   YLEN,PCR
383  22BB 34  04                   PSHS  B
384  22BD 34  10           PUT2    PSHS  X
385  22BF E6  8D 002E              LDB   XLEN,PCR
386  22C3 7D  D409                 TST   VRAM
387  22C6 A6  A0           PUT3    LDA   ,Y+
388  22C8 7D  D390                 TST   FUNC
389  22CB 27  01                   BEQ   PUT4       if PSET then jump
390  22CD 4F                       CLRA             if PRESET
391  22CE A7  80           PUT4    STA   ,X+
392  22D0 5A                       DECB
393  22D1 26  F3                   BNE   PUT3
394  22D3 B7  D409                 STA   VRAM
395  22D6 35  10                   PULS  X
396  22D8 30  88 50                LEAX  80,X       next line
397  22DB 6A  E4                   DEC   ,S
398  22DD 26  DE                   BNE   PUT2
399  22DF 35  04                   PULS  B
400  22E1 35  10                   PULS  X
401  22E3 30  89 4000              LEAX  $4000,X    next plane
402  22E7 6A  E4                   DEC   ,S
403  22E9 26  CA                   BNE   PUT1
404  22EB 35  04                   PULS  B
405
406  22ED B7  D40D                 STA   INSLED
407  22F0 39                       RTS              end of command
408
409  22F1                  XLEN    RMB   1
410  22F2                  YLEN    RMB   1
411
412  22F3 0006             PATBL   FDB   PAT0-PATBL
413  22F5 0020                     FDB   PAT1-PATBL
414  22F7 00B2                     FDB   PAT2-PATBL
415
416  22F9 01 08            PAT0    FCB   1,8
417  22FB 18                       FCB   %00011000 BLUE
418  22FC 18                       FCB   %00011000
419  22FD 18                       FCB   %00011000
420  22FE 18                       FCB   %00011000
421  22FF 18                       FCB   %00011000
422  2300 18                       FCB   %00011000
```

```
423    2301 18                FCB    %00011000
424    2302 18                FCB    %00011000
425
426    2303 18                FCB    %00011000 RED
427    2304 18                FCB    %00011000
428    2305 18                FCB    %00011000
429    2306 18                FCB    %00011000
430    2307 18                FCB    %00011000
431    2308 18                FCB    %00011000
432    2309 18                FCB    %00011000
433    230A 18                FCB    %00011000
434
435    230B 18                FCB    %00011000 GREEN
436    230C 18                FCB    %00011000
437    230D 18                FCB    %00011000
438    230E 18                FCB    %00011000
439    230F 18                FCB    %00011000
440    2310 18                FCB    %00011000
441    2311 18                FCB    %00011000
442    2312 18                FCB    %00011000
443
444    2313 03 10      PAT1   FCB    3,16
445    2315 00 18 00          FCB    %00000000,%00011000,%00000000 BLUE
446    2318 00 18 00          FCB    %00000000,%00011000,%00000000
447    231B 00 18 00          FCB    %00000000,%00011000,%00000000
448    231E 00 24 00          FCB    %00000000,%00100100,%00000000
449    2321 00 7E 00          FCB    %00000000,%01111110,%00000000
450    2324 18 66 18          FCB    %00011000,%01100110,%00011000
451    2327 3C 66 3C          FCB    %00111100,%01100110,%00111100
452    232A 7E 66 7E          FCB    %01111110,%01100110,%01111110
453    232D FF E7 FF          FCB    %11111111,%11100111,%11111111
454    2330 FF E7 FF          FCB    %11111111,%11100111,%11111111
455    2333 FF FF FF          FCB    %11111111,%11111111,%11111111
456    2336 FF 00 FF          FCB    %11111111,%00000000,%11111111
457    2339 E7 00 E7          FCB    %11100111,%00000000,%11100111
458    233C C3 00 C3          FCB    %11000011,%00000000,%11000011
459    233F 81 00 81          FCB    %10000001,%00000000,%10000001
460    2342 81 00 81          FCB    %10000001,%00000000,%10000001
461
462    2345 00 18 00          FCB    %00000000,%00011000,%00000000 RED
463    2348 00 18 00          FCB    %00000000,%00011000,%00000000
464    234B 00 18 00          FCB    %00000000,%00011000,%00000000
465    234E 00 3C 00          FCB    %00000000,%00111100,%00000000
466    2351 00 7E 00          FCB    %00000000,%01111110,%00000000
467    2354 18 66 18          FCB    %00011000,%01100110,%00011000
468    2357 24 42 24          FCB    %00100100,%01000010,%00100100
469    235A 42 42 42          FCB    %01000010,%01000010,%01000010
470    235D 81 81 81          FCB    %10000001,%10000001,%10000001
471    2360 81 81 81          FCB    %10000001,%10000001,%10000001
472    2363 81 FF 81          FCB    %10000001,%11111111,%10000001
473    2366 99 00 99          FCB    %10011001,%00000000,%10011001
474    2369 BD 00 BD          FCB    %10111101,%00000000,%10111101
475    236C FF 00 FF          FCB    %11111111,%00000000,%11111111
476    236F BD 00 BD          FCB    %10111101,%00000000,%10111101
477    2372 99 00 99          FCB    %10011001,%00000000,%10011001
478
479    2375 00 18 00          FCB    %00000000,%00011000,%00000000 GREEN
480    2378 00 18 00          FCB    %00000000,%00011000,%00000000
481    237B 00 18 00          FCB    %00000000,%00011000,%00000000
482    237E 00 3C 00          FCB    %00000000,%00111100,%00000000
483    2381 00 7E 00          FCB    %00000000,%01111110,%00000000
484    2384 18 7E 18          FCB    %00011000,%01111110,%00011000
485    2387 24 7E 24          FCB    %00100100,%01111110,%00100100
486    238A 66 7E 66          FCB    %01100110,%01111110,%01100110
487    238D A5 FF A5          FCB    %10100101,%11111111,%10100101
488    2390 A5 FF A5          FCB    %10100101,%11111111,%10100101
489    2393 A5 FF A5          FCB    %10100101,%11111111,%10100101
490    2396 BD 00 BD          FCB    %10111101,%00000000,%10111101
491    2399 A5 00 A5          FCB    %10100101,%00000000,%10100101
492    239C C3 00 C3          FCB    %11000011,%00000000,%11000011
```

```
563   243A A7    88 12            STA    $11+1,X    set X
564   243D 16    FC69             LBRA   TRANS
565
566   2440 13              CANPTN FCB    CANP2-CANP1
567   2441 00 00 3F        CANP1  FCB    0,0,$3F
568   2444 50 55 54 43            FCC    /PUTCANON/
569   244C 93 C0 00 90            FCB    $93,$C0,$00,$90
570   2450 01 00 00 17            FCB    1,0,0,23
571                 2454   CANP2  EQU    *
572                        *
573                        * MISSILE MOVE ROUTINE
574                        *
575                 2454   MSIL   EQU    *
576   2454 B6    FD04             LDA    BREAK      read fire key
577   2457 84    02               ANDA   #$02
578   2459 27    06               BEQ    MSIL1
579   245B A7    8D FC2E          STA    KEYFLG,PCR set keyflag
580   245F 20    35               BRA    MSIL4
581   2461 6D    8D FC28   MSIL1  TST    KEYFLG,PCR test keyflag
582   2465 27    2F               BEQ    MSIL4      can't fire ?
583   2467 6F    8D FC22          CLR    KEYFLG,PCR reset keyflag
584   246B 10AE  8D FC15          LDY    MISBUF,PCR load first missile
585   2470 E6    8D FB8F          LDB    MISILN,PCR
586   2474 6D    A4        MSIL2  TST    ,Y         live missile ?,
587   2476 27    07               BEQ    MSIL3
588   2478 31    23               LEAY   3,Y        try next mis.
589   247A 5A                     DECB
590   247B 26    F7               BNE    MSIL2
591   247D 20    17               BRA    MSIL4
592   247F C6    16        MSIL3  LDB    #22        set missile pos.
593   2481 A6    8D FBFF          LDA    CANX,PCR
594   2485 4C                     INCA
595   2486 ED    21               STD    1,Y
596   2488 E7    A4               STB    ,Y         set missile flag
597   248A CC    07F7             LDD    #$07F7     output sound
598   248D 17    FC2D             LBSR   PSG
599   2490 CC    0D09             LDD    #$0D09
600   2493 17    FC27             LBSR   PSG
601   2496 10AE  8D FBEA   MSIL4  LDY    MISBUF,PCR load first missile
602   249B E6    8D FB64          LDB    MISILN,PCR
603   249F 34    04               PSHS   B
604   24A1 6D    A4        MSIL5  TST    ,Y         check missile flag
605   24A3 27    0E               BEQ    MSIL6
606   24A5 86    01               LDA    #1         clear missile
607   24A7 8D    16               BSR    MISPUT
608   24A9 E6    22               LDB    2,Y        next position
609   24AB 5A                     DECB
610   24AC 2B    0D               BMI    MSIL7      output screen ?
611   24AE E7    22               STB    2,Y
612   24B0 4F                     CLRA              display missile
613   24B1 8D    0C               BSR    MISPUT
614   24B3 31    23        MSIL6  LEAY   3,Y        next missile
615   24B5 6A    E4               DEC    ,S
616   24B7 26    E8               BNE    MSIL5
617   24B9 35    84               PULS   B,PC
618   24BB 6F    A4        MSIL7  CLR    ,Y         missile dead
619   24BD 20    F4               BRA    MSIL6
620
621   24BF 30    8D 000B   MISPUT LEAX   MISPTN,PCR
622   24C3 A7    88 11            STA    $10+1,X    set func code
623   24C6 EC    21               LDD    1,Y
624   24C8 ED    88 12            STD    $11+1,X    set X,Y
625   24CB 16    FBDB             LBRA   TRANS
626
627   24CE 13              MISPTN FCB    MISP2-MISP1
628   24CF 00 00 3F        MISP1  FCB    0,0,$3F
629   24D2 50 55 54 4D            FCC    /PUTMISLE/
630   24DA 93 C0 00 90            FCB    $93,$C0,$00,$90
631   24DE 00 00 00 00            FCB    0,0,0,0
632                 24E2   MISP2  EQU    *
```

```
633
634                              *
635                              * ENEMY MOVE ROUTINE
636                              *
637                    24E2      ENEMY    EQU    *
638  24E2 E6    8D FB1C                   LDB    ENEMYN,PCR select first enemy
639  24E6 10AE  8D FB9C                   LDY    EMYBUF,PCR
640  24EB 34    04                        PSHS   B
641  24ED 8D    08                ENEMY1  BSR    ENEMYX     each enemy move
642  24EF 31    24                        LEAY   4,Y        select next enemy
643  24F1 6A    E4                        DEC    ,S
644  24F3 26    F8                        BNE    ENEMY1
645  24F5 35    84                        PULS   B,PC
646
647                    24F7      ENEMYX   EQU    *
648  24F7 6D    A4                        TST    ,Y         dead enemy ?
649  24F9 2B    47                        BMI    EMY5
650  24FB 86    01                        LDA    #1         erase enemy
651  24FD 17    0089                      LBSR   EMYPUT
652  2500 EC    B8 02             EMY1    LDD    [2,Y]      load delta data
653  2503 27    48                        BEQ    EMY6
654  2505 AB    A4                        ADDA   ,Y         culc next position
655  2507 EB    21                        ADDB   1,Y
656  2509 4D                              TSTA              test Xpos
657  250A 2B    36                        BMI    EMY5
658  250C 81    4E                        CMPA   #78
659  250E 22    32                        BHI    EMY5
660  2510 5D                              TSTB              test Ypos
661  2511 2B    2F                        BMI    EMY5
662  2513 C1    17                        CMPB   #23
663  2515 22    2B                        BHI    EMY5
664  2517 27    04                        BEQ    EMY2
665  2519 C1    16                        CMPB   #22
666  251B 26    19                        BNE    EMY4
667  251D 34    02                EMY2    PSHS   A          Y=22 or 23
668  251F 4C                              INCA
669  2520 A1    8D FB60                   CMPA   CANX,PCR
670  2524 25    0E                        BLO    EMY3
671  2526 80    03                        SUBA   #3
672  2528 A1    8D FB58                   CMPA   CANX,PCR
673  252C 22    06                        BHI    EMY3
674  252E 86    01                        LDA    #1
675  2530 A7    8D FB4D                   STA    SIGNAL,PCR cannon dead
676  2534 35    02                EMY3    PULS   A
677  2536 ED    A4                EMY4    STD    ,Y         set new pos.
678  2538 EC    22                        LDD    2,Y        select new delta
679  253A C3    0002                      ADDD   #2
680  253D ED    22                        STD    2,Y
681  253F 4F                              CLRA
682  2540 20    47                        BRA    EMYPUT
683
684  2542 8D    1D                EMY5    BSR    RND        random no. 8..71
685  2544 C4    3F                        ANDB   #64-1
686  2546 CB    08                        ADDB   #8
687  2548 4F                              CLRA
688  2549 1E    89                        EXG    A,B
689  254B ED    A4                        STD    ,Y         set new position
690  254D 8D    12                EMY6    BSR    RND        select new delta-pattern
691  254F 54                              LSRB
692  2550 54                              LSRB
693  2551 54                              LSRB
694  2552 C4    0F                        ANDB   #PATNN-1
695  2554 58                              ASLB
696  2555 30    8D 0173                   LEAX   DTABLE,PCR load delta addr.
697  2559 EC    85                        LDD    B,X
698  255B 30    8B                        LEAX   D,X
699  255D AF    22                        STX    2,Y
700  255F 20    9F                        BRA    EMY1
701
702                              *
```

```
703                            * RND ROUTINE
704                            *
705   2561 34    12            RND      PSHS   A,X
706   2563 30    8D FB22                LEAX   RNDWRK,PCR Xi+1=Xi*RNDC+13(mod2^16)
707   2567 A6    01                     LDA    1,X
708   2569 C6    7B                     LDB    #RNDC&$FF
709   256B 3D                           MUL
710   256C ED    02                     STD    2,X
711   256E A6    01                     LDA    1,X
712   2570 C6    00                     LDB    #RNDC/256
713   2572 3D                           MUL
714   2573 E3    01                     ADDD   1,X
715   2575 ED    01                     STD    1,X
716   2577 A6    84                     LDA    ,X
717   2579 C6    7B                     LDB    #RNDC&$FF
718   257B 3D                           MUL
719   257C E3    01                     ADDD   1,X
720   257E ED    01                     STD    1,X
721   2580 EC    02                     LDD    2,X
722   2582 C3    000D                   ADDD   #13
723   2585 ED    84                     STD    ,X
724   2587 35    92                     PULS   A,X,PC     return RND in Breg
725
726                            *
727                            * ENEMY DISPLAY
728                            *
729   2589 30    8D 000B       EMYPUT   LEAX   EMYPTN,PCR
730   258D A7    88 11                  STA    $10+1,X    set func code
731   2590 EC    A4                     LDD    ,Y
732   2592 ED    88 12                  STD    $11+1,X    set X,Y
733   2595 16    FB11                   LBRA   TRANS
734
735   2598 13                  EMYPTN   FCB    EMYP2-EMYP1
736   2599 00 00 3F            EMYP1    FCB    0,0,$3F
737   259C 45 4E 45 4D                  FCC    /ENEMYPUT/
738   25A4 93 C0 00 90                  FCB    $93,$C0,$00,$90
739   25A8 02 00 00 00                  FCB    2,0,0,0
740             25AC           EMYP2    EQU    *
741
742                            *
743                            * CHECK ROUTINE
744                            *
745             25AC           CHECK    EQU    *
746   25AC 10AE 8D FAD6                 LDY    EMYBUF,PCR Yreg=enemy buf.
747   25B1 E6    8D FA4D                LDB    ENEMYN,PCR
748   25B5 34    04                     PSHS   B
749   25B7 EE    8D FACA       CHECK1   LDU    MISBUF,PCR Ureg=missile buf
750   25BB E6    8D FA44                LDB    MISILN,PCR
751   25BF 34    04                     PSHS   B
752   25C1 EC    A4            CHECK2   LDD    ,Y         check position
753   25C3 6D    C4                     TST    ,U
754   25C5 27    43                     BEQ    CHECK4
755   25C7 10A3 41                      CMPD   1,U
756   25CA 27    12                     BEQ    CHECK3
757   25CC 4C                           INCA
758   25CD 10A3 41                      CMPD   1,U
759   25D0 27    0C                     BEQ    CHECK3
760   25D2 5C                           INCB
761   25D3 10A3 41                      CMPD   1,U
762   25D6 27    06                     BEQ    CHECK3
763   25D8 4A                           DECA
764   25D9 10A3 41                      CMPD   1,U
765   25DC 26    2C                     BNE    CHECK4
766   25DE EC    8D FAA0       CHECK3   LDD    SCORE,PCR bomb !!
767   25E2 C3    000A                   ADDD   #10        score = score +10
768   25E5 ED    8D FA99                STD    SCORE,PCR
769   25E9 CC    07F9                   LDD    #$07F9     bomb sound
770   25EC 17    FACE                   LBSR   PSG
771   25EF CC    0D09                   LDD    #$0D09
772   25F2 17    FAC8                   LBSR   PSG
```

```
773   25F5 1E   23              EXG   Y,U
774   25F7 86   01              LDA   #1          erase missile
775   25F9 17   FEC3            LBSR  MISPUT
776   25FC 1E   23              EXG   Y,U
777   25FE 6F   C4              CLR   ,U
778   2600 86   01              LDA   #1          erase enemy
779  >2602 17   FF84            LBSR  EMYPUT
780   2605 CC   COCO            LDD   #$COCO      point out of screen
781   2608 ED   A4              STD   ,Y
782   260A 33   43     CHECK4   LEAU  3,U         select next missile
783   260C 6A   E4              DEC   ,S
784   260E 26   B1              BNE   CHECK2
785   2610 35   04              PULS  B
786   2612 31   24              LEAY  4,Y         select next enemy
787   2614 6A   E4              DEC   ,S
788   2616 26   9F              BNE   CHECK1
789   2618 35   84              PULS  B,PC
790
791                        *
792                        * STAR MOVE
793                        *
794   261A 86   01         STAR     LDA   #1          erase stars
795   261C 8D   1F                  BSR   STRPUT
796   261E 108E 0014       STAR0    LDY   #20
797   2622 30   8D 0030             LEAX  STRPTN+7,PCR
798   2626 EC   84         STAR1    LDD   ,X          star down
799   2628 C3   0004                ADDD  #4
800   262B 1083 00C8                CMPD  #200
801   262F 25   03                  BLO   STAR2
802   2631 83   00C8                SUBD  #200
803   2634 ED   84         STAR2    STD   ,X
804   2636 30   06                  LEAX  6,X         next star
805   2638 31   3F                  LEAY  -1,Y
806   263A 26   EA                  BNE   STAR1
807   263C 4F                       CLRA              display stars
808   263D 30   8D 000E    STRPUT   LEAX  STRPTN,PCR
809   2641 31   0A                  LEAY  10,X
810   2643 C6   14                  LDB   #20
811   2645 A7   A4         STRPT1   STA   ,Y          set function code
812   2647 31   26                  LEAY  6,Y
813   2649 5A                       DECB
814   264A 26   F9                  BNE   STRPT1
815   264C 16   FA5A                LBRA  TRANS
816                        *
817                        * STAR PATTERN
818                        *
819   264F 7C              STRPTN   FCB   STRP2-STRP1
820   2650 00 00 17 14     STRP1    FCB   0,0,$17,20
821   2654 002E 0000                FDB   046,000,$0104,310,100,$0204
822   2660 009B 0014                FDB   155,020,$0304,415,130,$0404
823   266C 00EB 0028                FDB   235,040,$0504,512,150,$0604
824   2678 0183 003C                FDB   387,060,$0704,635,170,$0104
825   2684 01D0 0050                FDB   464,080,$0204,024,190,$0304
826   2690 0237 0064                FDB   567,100,$0404,155,010,$0504
827   269C 0271 0078                FDB   625,120,$0604,278,030,$0704
828   26A8 004A 008C                FDB   074,140,$0104,397,050,$0204
829   26B4 00BB 00A0                FDB   187,160,$0304,488,070,$0404
830   26C0 00F4 00B4                FDB   244,180,$0504,555,090,$0604
831                 26CC   STRP2    EQU   *
832
833                        *
834                        * DATA OF ENEMY MOVE
835                        *
836                 0010   PATNN    EQU   16
837   26CC 0020            DTABLE   FDB   DELTA1-DTABLE
838   26CE 002A                     FDB   DELTA2-DTABLE
839   26D0 0034                     FDB   DELTA3-DTABLE
840   26D2 003E                     FDB   DELTA4-DTABLE
841   26D4 0048                     FDB   DELTA5-DTABLE
842   26D6 0052                     FDB   DELTA6-DTABLE
```

```
843    26D8 005C                     FDB   DELTA7-DTABLE
844    26DA 0066                     FDB   DELTA8-DTABLE
845    26DC 0070                     FDB   DELTA9-DTABLE
846    26DE 007A                     FDB   DELTA0-DTABLE
847    26E0 0020                     FDB   DELTA1-DTABLE
848    26E2 002A                     FDB   DELTA2-DTABLE
849    26E4 0034                     FDB   DELTA3-DTABLE
850    26E6 003E                     FDB   DELTA4-DTABLE
851    26E8 0048                     FDB   DELTA5-DTABLE
852    26EA 0052                     FDB   DELTA6-DTABLE
853
854    26EC 0001 0001       DELTA1   FDB   $0001,$0001,$0001,$0001,0
855    26F6 0002 0002       DELTA2   FDB   $0002,$0002,$0002,$0002,0
856    2700 FF01 FF01       DELTA3   FDB   $FF01,$FF01,$FF01,$FF01,0
857    270A 0101 0101       DELTA4   FDB   $0101,$0101,$0101,$0101,0
858    2714 FF00 FF00       DELTA5   FDB   $FF00,$FF00,$FF00,$FF00,0
859    271E 0100 0100       DELTA6   FDB   $0100,$0100,$0100,$0100,0
860    2728 00FF 00FF       DELTA7   FDB   $00FF,$00FF,$00FF,$00FF,0
861    2732 00FE 00FE       DELTA8   FDB   $00FE,$00FE,$00FE,$00FE,0
862    273C FFFF FFFF       DELTA9   FDB   $FFFF,$FFFF,$FFFF,$FFFF,0
863    2746 01FF 01FF       DELTA0   FDB   $01FF,$01FF,$01FF,$01FF,0
864
865    2750 000E            MSGTBL   FDB   MES0-MSGTBL
866    2752 007A                     FDB   MES1-MSGTBL
867    2754 00C9                     FDB   MES2-MSGTBL
868    2756 0102                     FDB   MES3-MSGTBL
869    2758 0155                     FDB   MES4-MSGTBL
870    275A 01BF                     FDB   MES5-MSGTBL
871    275C 01F5                     FDB   MES6-MSGTBL
872
873                    0012 LOCATE   EQU   $12
874                    000C CLS      EQU   $0C
875    275E 6B              MES0     FCB   MES1-MES0-1
876    275F 12 14 04                 FCB   LOCATE,20,4
877    2762 3D 3D 3D 3D              FCC   "===== PART ][ / Version 1.00 ====="
878    2784 12 08 06                 FCB   LOCATE,8,6
879    2787 BA C9 20 B9              FCC   "ｺﾉ ｹﾞｰﾑ ﾊ ｢FM-7/FM-NEW7/FM-77 "
880    27A5 CF BC DD BA              FCC   "ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ ﾏﾆｭｱﾙ｣ ﾉ ﾀﾒﾉ"
881    27BD 20 CC DF DB              FCC   " ﾌﾟﾛｸﾞﾗﾑ ﾃﾞｽ｡"
882    27CA 4E              MES1     FCB   MES2-MES1-1
883    27CB 12 14 08                 FCB   LOCATE,20,8
884    27CE 3C 3C 20 B1              FCC   "<< ｱ ｿ ﾋﾞ ｶ ﾀ >>"
885    27DE 12 14 0A                 FCB   LOCATE,20,10
886    27E1 20 B3 B4 B6              FCC   " ｳｴｶﾗ ｵﾘﾃｸﾙ ｴｲﾘｱﾝ ｦ ｷｬﾉﾝ ﾉ "
887    27FC CA B6 B2 20              FCC   "ﾊｶｲ ｺｳｾﾝ ﾃﾞ ｹﾞｷﾊ ｼﾃ ｸﾀﾞｻｲ｡"
888    2816 12 14 0C                 FCB   LOCATE,20,12
889    2819 38              MES2     FCB   MES3-MES2-1
890    281A 3C 3C 20 B7              FCC   "<< ｷ - ﾉ ｿ ｳ ｻ >>"
891    282B 12 12 0D                 FCB   LOCATE,18,13
892    282E 20 20 20 20              FCC   "     ┌─┐                    "
893    2842 20 20 20 20              FCC   "                    ┌─┐"
894    2852 52              MES3     FCB   MES4-MES3-1
895    2853 12 12 0E                 FCB   LOCATE,18,14
896    2856 3C 2D 2D 20              FCC   "<-- |4| LEFT MOVE  /"
897    286A 20 20 52 49              FCC   "  RIGHT MOVE |6| -->"
898    287E 12 12 0F                 FCB   LOCATE,18,15
899    2881 20 20 20 20              FCC   "     └─┘                    "
900    2895 20 20 20 20              FCC   "                    └─┘"
901    28A5 69              MES4     FCB   MES5-MES4-1
902    28A6 12 12 10                 FCB   LOCATE,18,16
903    28A9 20 20 20 98              FCC   "    ┌──────────┐           ┌"
904    28BD 95 95 95 95              FCC   "───────────┐"
905    28C5 12 12 11                 FCB   LOCATE,18,17
906    28C8 20 20 20 96              FCC   "    |BREAK| FIRE      !"
907    28DC 20 53 50 41              FCC   " SPACE | GAME START "
908    28F0 12 12 12                 FCB   LOCATE,18,18
909    28F3 20 20 20 9A              FCC   "    └──────────┘           └"
910    2907 95 95 95 95              FCC   "───────────┘"
911    290F 35              MES5     FCB   MES6-MES5-1
912    2910 12 14 14                 FCB   LOCATE,20,20
```

```
  913   2913 43 4F 50 59                 FCC      "COPYRIGHT (C) 1984 "
  914   2926 62 79 20 48                 FCC      "by H.NAKAMURA"
  915   2933 12 20 16                    FCB      LOCATE,32,22
  916   2936 48 49 54 20                 FCC      "HIT SPACE KEY !"
  917   2945 2A               MES6       FCB      MES7-MES6-1
  918   2946 12 23 0F                    FCB      LOCATE,35,15
  919   2949 47 41 4D 45                 FCC      "GAME  OVER"
  920   2953 12 1E 17                    FCB      LOCATE,30,23
  921   2956 52 45 50 4C                 FCC      "REPLAY ? (Y or N)"
  922   2967 12 23 12                    FCB      LOCATE,35,18
  923   296A 53 43 4F 52                 FCC      "SCORE="
  924               2970      MES7       EQU      *
  925
  926                         *
  927                         * end of program part
  928                         *
  929                         * buffer of enemy & missile from here
  930                         *
  931               2970      BUFFER     EQU      *
  932
  933                                    END      START

0 ERROR(S) DETECTED

SYMBOL TABLE:

BREAK   FD04    BUFFER  2970    CAN1    240B    CAN2    241E    CAN3    242E
CANNON  2407    CANP1   2441    CANP2   2454    CANPTN  2440    CANPUT  242F
CANX    2084    CHECK   25AC    CHECK1  25B7    CHECK2  25C1    CHECK3  25DE
CHECK4  260A    CLS     000C    CLSCOM  21CF    COFFE   2250    COFFS   224C
COUNT0  208E    COUNT1  208F    COUNT2  2090    COUNT3  2091    CUROFF  224B
DELTA0  2746    DELTA1  26EC    DELTA2  26F6    DELTA3  2700    DELTA4  270A
DELTA5  2714    DELTA6  271E    DELTA7  2728    DELTA8  2732    DELTA9  273C
DTABLE  26CC    EINKYC  20F7    EMY1    2500    EMY2    251D    EMY3    2534
EMY4    2536    EMY5    2542    EMY6    254D    EMYBUF  2087    EMYP1   2599
EMYP2   25AC    EMYPTN  2598    EMYPUT  2589    ENDPRO  2077    ENEMY   24E2
ENEMY1  24ED    ENEMYN  2002    ENEMYX  24F7    FUNC    D390    GAME    2004
GAMEX   2064    HALT    FD05    INIT    215F    INIT1   216B    INIT2   2189
INIT3   219A    INIT4   21B1    INKEYC  20F2    INSLED  D40D    INTM0   21F2
INTM1   2212    INTMSG  21EC    KEYBRD  FD01    KEYFLG  208D    KEYIN   20D6
LOCATE  0012    MES0    275E    MES1    27CA    MES2    2819    MES3    2852
MES4    28A5    MES5    290F    MES6    2945    MES7    2970    MISBUF  2085
MISILN  2003    MISP1   24CF    MISP2   24E2    MISPTN  24CE    MISPUT  24BF
ML1     202A    ML2     2039    ML3     204B    ML4     205C    MLOOP   2012
MSGTBL  2750    MSIL    2454    MSIL1   2461    MSIL2   2474    MSIL3   247F
MSIL4   2496    MSIL5   24A1    MSIL6   24B3    MSIL7   24BB    NUMBUF  2155
NUMTB   214B    PAT0    22F9    PAT1    2313    PAT2    23A5    PATBL   22F3
PATNN   0010    PATRN   D38F    POSX    D391    POSY    D392    PRINT   20F7
PRINT1  210E    PSG     20BD    PSGC    FD0D    PSGD    FD0E    PSGDAT  21D6
PUT1    22B5    PUT2    22BD    PUT3    22C6    PUT4    22CE    RND     2561
RNDC    007B    RNDWRK  2089    RUN     20A5    SCOM1   221C    SCOM2   2230
SCORE   2082    SCOUT   211A    SCOUT1  2126    SCOUT2  212E    SCRNI   21C3
SIGNAL  2081    SINKYC  20F3    SRAM    FC80    STAR    261A    STAR0   261E
STAR1   2626    STAR2   2634    START   2000    STOP    2092    STOP1   2094
STOP2   209E    STRP1   2650    STRP2   26CC    STRPT1  2645    STRPTN  264F
STRPUT  263D    SUBPRE  2407    SUBPRO  228D    SUBSB1  2243    SUBSUB  2230
SYMCOM  221B    TESTCE  228D    TESTCS  227F    TOSUB   2250    TOSUB1  225E
TOSUB2  2273    TRANS   20A9    TRANS1  20B2    VRAM    D409    XLEN    22F1
YLEN    22F2
```

〔図E－18 ダンプリスト〕

```
2000: 20 02 06 0A 17 01 58 30 8D 00 83 CC 01 01 ED 84 :21   .....X0■.▂7ˋ.O■
2010: ED 02 6A 8D 00 78 26 12 86 06 A7 8D 00 70 17 04 :E1   O.j■.x&.■.ｧ■.p..
2020: C1 6D 8D 00 5C 26 35 17 05 82 6A 8D 00 61 26 09 :97   ﾁm■.¥&5..▂j■.a&.
2030: 86 03 A7 8D 00 59 17 03 CE 6A 8D 00 53 26 0C 86 :00   ■.ｧ■.Y..ﾎj■.S&.■
2040: 02 A7 8D 00 4B 17 04 0C 17 05 61 6A 8D 00 42 26 :84   .ｧ■.K.....aj■.B&
```

```
2050: C1 B6 0A A7 8D 00 3A 17 05 C0 20 B6 C6 06 17 00 :54   チ■.ァ■.:..タ カニ...
2060: 96 17 00 B6 17 00 6F 81 59 27 99 81 79 27 95 81 :BA   |..カ..o_Y'ヶ_y'ー_
2070: 4E 27 04 81 6E 26 ED 30 BD 01 54 17 00 2B 1C AF :9A   N'._n&O0■.T..+.ッ
2080: 39 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 :3A   9...............
2090: 00 00 34 02 B6 FD 05 2B FB 86 80 B7 FD 05 B6 FD :86   ..4.カ人.+町■_キ人.カ人
20A0: 05 2A FB 35 82 7F FD 05 39 34 56 8D E5 E6 80 CE :CB   .*町5_.人.94V■◣◥_ホ
20B0: FC 80 A6 80 A7 C0 5A 26 F9 8D EA 35 D6 34 06 B7 :F5   木_ヲ_ァタZ&ホ■◆5ヨ4.キ
20C0: FD 0E 86 03 B7 FD 0D 7F FD 0D F7 FD 0E 4A B7 FD :DE   人.■.キ人..人.叙人.Jキ人
20D0: 0D 7F FD 0D 35 86 34 14 30 8D 00 16 17 FF CA 17 :63   ..人.5■4.0■... ハ.
20E0: FF B0 B6 FC 83 F6 FC 80 CA 80 F7 FC 80 17 FF B5 :DE    ーカ木_分木_ハ_叙木_. オ
20F0: 35 94 04 00 00 29 03 34 56 17 FF 96 CE FC 82 86 :01   5¯...).4V. |ホ木_■
2100: 03 A7 C0 58 30 8D 06 48 EC 85 30 8B E6 84 A6 80 :89   .ァタX0■.H●■0|◥_ヲ_
2110: A7 C0 5A 2A F9 17 FF 8D 35 D6 33 8D 00 2D 30 8D :3C   ァタZ*ホ. ■5ヨ3■.-0■
2120: 00 38 86 05 34 02 86 2F 34 02 EC 8D FF 54 6C E4 :00   .8■.4.■/4.■ Tl◢
2130: A3 C4 24 FA E3 C1 ED 8D FF 48 35 02 A7 80 6A E4 :96   」ト$区ヨチ○■ H5.ァ_j◢
2140: 26 E4 35 02 30 8D 00 0D 16 FF 5E 27 10 03 E8 00 :A0   &◢5.0■... ^'..♠.
2150: 64 00 0A 00 01 09 00 00 03 05 30 30 30 30 30 1A :8A   d..........00000.
2160: 50 30 8D 00 5E 17 FF 41 17 00 81 17 FF 68 81 20 :79   P0■.^. A..._. h_
2170: 26 F9 6F 8D FF 0B CC 00 00 ED 8D FF 05 30 8D 07 :33   &ホo■ .フ..○■ .0■.
2180: EF AF 8D FF 00 E6 8D FE 7A 6F 80 6F 80 6F 80 5A :3C   ＼ッ■ .◥■※zo_o_o_Z
2190: 26 F7 AF 8D FE F1 E6 8D FE 68 6F 84 6F 01 AF 02 :35   &叙ッ■※円◥■※ho■o.ッ.
21A0: 30 04 5A 26 F5 86 28 A7 8D FE D9 30 8D 00 27 C6 :0C   0.Z&時■(ァ■※ル0■.'ニ
21B0: 0B EC 81 17 FF 07 5A 26 F8 30 8D 00 12 17 FE E9 :DA   .●_. .Z&〒0■...※♥
21C0: 16 00 6D 0B 00 00 01 00 50 19 00 19 00 01 00 06 :18   ..m.....P.......
21D0: 00 00 0D 00 00 07 07 F9 02 00 03 02 04 00 05 04 :28   ........ホ........
21E0: 06 00 08 10 09 10 0A 10 0B 00 0C 10 C6 03 30 8D :FE   ..............ニ.0■
21F0: 00 29 34 04 E7 0C CB A0 E7 0A 17 FE AC 35 04 5A :04   .)4.◤.ヒ ◤..※ャ5.Z
2200: 26 F0 6A 0C 6A 0A 86 05 A7 04 17 FE 9C 86 01 A7 :15   &×j.j.■.ァ..※◢■.ァ
2210: 04 5F 17 FE E2 5C C1 06 26 F8 39 14 00 00 19 01 :02   ._.※ヰ¥チ.&〒9.....
2220: 00 00 05 03 00 A0 00 00 07 5A 49 47 20 5A 41 50 :A4   ...... ...ZIG ZAP
2230: 30 8D 00 17 17 FE 72 10 8E C0 00 30 8D 00 4E C6 :8A   0■...※r.■タ.0■.Nニ
2240: 06 34 04 8D 0B 6A E4 26 FA 35 84 04 00 00 0C 1E :2B   .4.■.j◢&区5■.....
2250: 34 10 17 FE 3D 30 8D 00 26 CE FC 80 C6 0E A6 80 :BD   4..※=0■.&ホ木_ニ.ヲ_
2260: A7 C0 5A 26 F9 10 AF C1 CC 00 40 ED C1 86 90 A7 :D7   ァタZ&ホ.ッチフ.@○チ■┴ァ
2270: C0 35 10 A6 80 A7 C0 31 21 5A 26 F7 16 FE 26 00 :95   タ5.ヲ_ァタ1!Z&叙.※&.
2280: 00 3F 53 55 42 43 50 55 50 52 91 D3 93 B6 D4 0D :41   .?SUBCPUPRーモトカヤ.
2290: F6 D3 8F 58 31 8D 00 5B EC A5 31 AB B6 D3 92 C6 :17   分モ+X1■.[●・1ォカモ┤ニ
22A0: 08 3D 86 50 3D 1F 01 F6 D3 91 3A EC A1 ED 8D 00 :13   .=■P=..分モ┳:●。○■.
22B0: 40 C6 03 34 04 34 10 E6 8D 00 37 34 04 34 10 E6 :91   @ニ.4.4.◥■.74.4.◥
22C0: 8D 00 2E 7D D4 09 A6 A0 7D D3 90 27 01 4F A7 80 :D9   ■..}ヤ.ヲ }モ┴'.Oァ_
22D0: 5A 26 F3 B7 D4 09 35 10 30 8B 50 6A E4 26 DE 35 :DB   Z&月キヤ.5.0|Pj◢&゛5
22E0: 04 35 10 30 89 40 00 6A E4 26 CA 35 04 B7 D4 0D :51   .5.0| @.j◢&ハ5.キヤ.
22F0: 39 00 00 00 06 00 20 00 B2 01 08 18 18 18 18 18 :92   9..... .イ.......
2300: 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 :80   ................
2310: 18 18 18 03 10 00 18 00 00 18 00 00 18 00 00 24 :C7   ...............$
2320: 00 00 7E 00 18 66 18 3C 66 3C 7E 66 7E FF E7 FF :39   ..~..f.<f<~f~ ◤
2330: FF E7 FF FF FF FF FF 00 FF E7 00 E7 C3 00 C3 81 :B5    ◤       . ◤.◤テ.テ_
2340: 00 81 81 00 81 00 18 00 00 18 00 00 18 00 00 3C :07   .__._..........<
2350: 00 00 7E 00 18 66 18 24 42 24 42 42 42 81 81 81 :E7   ..~..f.$B$BBB___
2360: 81 81 81 81 FF 81 99 00 99 BD 00 BD FF 00 FF BD :EB   ____ _ヿ.ヿス.ス . ス
2370: 00 BD 99 00 99 00 18 00 00 18 00 00 18 00 00 3C :73   .スヿ.ヿ..........<
2380: 00 00 7E 00 18 7E 18 24 7E 24 66 7E 66 A5 FF A5 :85   ..~..~.$~$f~f・ ・
2390: A5 FF A5 A5 FF A5 BD 00 BD A5 00 A5 C3 00 C3 81 :5D   ・ ・・ ・ス.ス・.・テ.テ_
23A0: 00 81 81 00 81 02 10 FF FF 70 0E 38 1C 1C 38 0E :C7   .__._..  p.8..8.
23B0: 70 07 E0 03 C0 00 0C 00 0C 03 C0 07 E0 0E 70 1C :76   p.=.タ.....タ.=.p.
23C0: 38 38 1C 70 0E FF FF 00 00 70 0E 38 1C 1C 38 0E :3C   88.p.  ..p.8..8.
23D0: 70 07 E0 03 C0 30 00 30 00 03 C0 07 E0 0E 70 1C :BE   p.=.タ0.0..タ.=.p.
23E0: 38 38 1C 70 0E 00 00 00 00 00 00 00 00 00 00 00 :0A   88.p...........
23F0: 00 00 00 00 00 0F FC 0F FC 00 00 00 00 00 00 00 :16   .....木.木.......
2400: 00 00 00 00 00 00 00 86 01 8D 24 B6 FD 01 81 34 :A1   .......■.■$カ人._4
2410: 26 0C 6D 8D FC 6E 27 16 6A 8D FC 68 20 10 81 36 :15   &.m■木n'.j■木h ._6
2420: 26 0C E6 8D FC 5E C1 4D 24 04 6C 8D FC 56 4F 30 :FF   &.◥■木^チM$.l■木VO0
2430: 8D 00 0D A7 88 11 A6 8D FC 4A A7 88 12 16 FC 69 :0F   ■..ァ|.ヲ■木Jァ|..木i
2440: 13 00 00 3F 50 55 54 43 41 4E 4F 4E 93 C0 00 90 :9D   ...?PUTCANONトタ.┴
2450: 01 00 00 17 B6 FD 04 84 02 27 06 A7 8D FC 2E 20 :00   ....カ人.■.'.ァ■木.
2460: 35 6D 8D FC 28 27 2F 6F 8D FC 22 10 AE 8D FC 15 :1F   5m■木('/o■木".ョ■木.
2470: E6 8D FB 8F 6D A4 27 07 31 23 5A 26 F7 20 17 C6 :04   ◥■町+m、'.1#Z&叙 .ニ
2480: 16 A6 8D FB FF 4C ED 21 E7 A4 CC 07 F7 17 FC 2D :32   .ヲ■町 LO!◤、フ.叙.木-
2490: CC 0D 09 17 FC 27 10 AE 8D FB EA E6 8D FB 64 34 :52   フ...木'.ョ■町◆◥■町d4
24A0: 04 6D A4 27 0E 86 01 8D 16 E6 22 5A 2B 0D E7 22 :17   .m、'.■.■.◥"Z+.◤"
```

```
24B0: 4F 8D 0C 31 23 6A E4 26 E8 35 84 6F A4 20 F4 30 :A8  O■.1#j◢&◆5▬o、 BO
24C0: 8D 00 0B A7 88 11 EC 21 ED 88 12 16 FB DB 13 00 :6B  ■..ｧ| .●!ｺ| ..町ﾛ..
24D0: 00 3F 50 55 54 4D 49 53 4C 45 93 C0 00 90 00 00 :95  .?PUTMISLE ﾄﾀ.┴..
24E0: 00 00 E6 8D FB 1C 10 AE 8D FB 9C 34 04 8D 08 31 :6A  ..◥町..ｮ■町／4.■.1
24F0: 24 6A E4 26 F8 35 84 6D A4 2B 47 86 01 17 00 89 :F3  $j◢&〒5▬m、+G■...|
2500: EC B8 02 27 48 AB A4 EB 21 4D 2B 36 81 4E 22 32 :41  ●ｸ.'Hｫ、♣!M+6_N"2
2510: 5D 2B 2F C1 17 22 2B 27 04 C1 16 26 19 34 02 4C :9F  ]+/ﾁ."+'.ﾁ.&.4.L
2520: A1 8D FB 60 25 0E 80 03 A1 8D FB 58 22 06 86 01 :6F  ｡■町`%.＿.｡■町X".■.
2530: A7 8D FB 4D 35 02 ED A4 EC 22 C3 00 02 ED 22 4F :75  ｧ■町M5.ｺ、●"ﾃ..ｺ"O
2540: 20 47 8D 1D C4 3F CB 08 4F 1E 89 ED A4 8D 12 54 :61   G■.ﾄ?ﾋ.O.|ｺ、■.T
2550: 54 54 C4 0F 58 30 8D 01 73 EC 85 30 8B AF 22 20 :21  TTﾄ.X0■.s●▬0■ｯ"
2560: 9F 34 12 30 8D FB 22 A6 01 C6 7B 3D ED 02 A6 01 :7A  ╱4.0■町"ｦ.ﾆ{=ｺ.ｦ.
2570: C6 00 3D E3 01 ED 01 A6 84 C6 7B 3D E3 01 ED 01 :4F  ﾆ.=ｺ.ｺ.ｦ▬ﾆ{=ｺ.ｺ.
2580: EC 02 C3 00 0D ED 84 35 92 30 8D 00 0B A7 88 11 :FE  ●.ﾃ..ｺ▬5┤0■..ｧ|.
2590: EC A4 ED 88 12 16 FB 11 13 00 00 3F 45 4E 45 4D :B0  ●、ｺ|..町....?ENEM
25A0: 59 50 55 54 93 C0 00 90 02 00 00 00 10 AE 8D FA :7C  YPUT ﾄﾀ.┴.....ｮ■■
25B0: D6 E6 8D FA 4D 34 04 EE 8D FA CA E6 8D FA 44 34 :EC  ﾖ◥■M4.／■■ﾊ◥■D4
25C0: 04 EC A4 6D C4 27 43 10 A3 41 27 12 4C 10 A3 41 :9C  .●、mﾄ'C.｣A'.L.｣A
25D0: 27 0C 5C 10 A3 41 27 06 4A 10 A3 41 26 2C EC 8D :B9  '.¥.｣A'.J.｣A&,●■
25E0: FA A0 C3 00 0A ED 8D FA 99 CC 07 F9 17 FA CE CC :EB  ■ ﾃ..ｺ■■┐ﾌ.ﾎ.■ﾎﾌ
25F0: 0D 09 17 FA C8 1E 23 86 01 17 FE C3 1E 23 6F C4 :03  ...■ﾈ.#■..※ﾃ.#oﾄ
2600: 86 01 17 FF 84 CC C0 C0 ED A4 33 43 6A E4 26 B1 :99  ■.. ▬ﾌﾀﾀｺ、3Cj◢&ｱ
2610: 35 04 31 24 6A E4 26 9F 35 84 86 01 8D 1F 10 8E :2B  5.1$j◢&╱5▬■.■..■
2620: 00 14 30 8D 00 30 EC 84 C3 00 04 10 83 00 C8 25 :B8  ..0■.0●▬ﾃ...▬.ﾈ%
2630: 03 83 00 C8 ED 84 30 06 31 3F 26 EA 4F 30 8D 00 :81  .▬.ﾈｺ▬0.1?&◆O0■.
2640: 0E 31 0A C6 14 A7 A4 31 26 5A 26 F9 16 FA 5A 7C :24  .1.ﾆ.ｧ、1&Z&ﾎ.■Z|
2650: 00 00 17 14 00 2E 00 00 01 04 01 36 00 64 02 04 :FF  ...........6.d..
2660: 00 9B 00 14 03 04 01 9F 00 82 04 04 00 EB 00 28 :F3  .┘.....╱.▬...♣.(
2670: 05 04 02 00 00 96 06 04 01 83 00 3C 07 04 02 7B :F3  .....|...▬.<...{
2680: 00 AA 01 04 01 D0 00 50 02 04 00 18 00 BE 03 04 :B3  .ｪ...ﾐ.P.....ｾ..
2690: 02 37 00 64 04 04 00 9B 00 0A 05 04 02 71 00 78 :3E  .7.d...┘.....q.x
26A0: 06 04 01 16 00 1E 07 04 00 4A 00 8C 01 04 01 8D :B3  .........J.■...■
26B0: 00 32 02 04 00 BB 00 A0 03 04 01 EB 00 46 04 04 :D1  .2...ｻ. ...♣.F..
26C0: 00 F4 00 B4 05 04 02 2B 00 5A 06 04 00 20 00 2A :8C  .■.ｴ...+.Z... .*
26D0: 00 34 00 3E 00 48 00 52 00 5C 00 66 00 70 00 7A :B8  .4.>.H.R.¥.f.p.z
26E0: 00 20 00 2A 00 34 00 3E 00 48 00 52 00 01 00 01 :58  . .*.4.>.H.R....
26F0: 00 01 00 01 00 00 00 02 00 02 00 02 00 02 00 00 :0A  ................
2700: FF 01 FF 01 FF 01 FF 01 00 00 01 01 01 01 01 01 :06   . . . .........
2710: 01 01 00 00 FF 00 FF 00 FF 00 FF 00 00 00 01 00 :FF  .... . . . ......
2720: 01 00 01 00 01 00 00 00 00 FF 00 FF 00 FF 00 FF :FF  ......... . . .
2730: 00 00 00 FE 00 FE 00 FE 00 FE 00 00 FF FF FF FF :F4  ...※.※.※.※..
2740: FF FF FF FF 00 00 01 FF 01 FF 01 FF 01 FF 00 00 :FC      ... . . . ..
2750: 00 0E 00 7A 00 C9 01 02 01 55 01 BF 01 F5 6B 12 :DD  ...z.ﾉ...U.ｿ.■k.
2760: 14 04 3D 3D 3D 3D 3D 20 50 41 52 54 20 5D 5B 20 :98  ..===== PART ][
2770: 2F 20 56 65 72 73 69 6F 6E 20 31 2E 30 30 20 3D :71  / Version 1.00 =
2780: 3D 3D 3D 3D 12 08 06 BA C9 20 B9 DE B0 D1 20 CA :B9  ====...ｺﾉ ｹﾞｰﾑ ﾊ
2790: 20 A2 46 4D 2D 37 2F 46 4D 2D 4E 45 57 37 2F 46 :3E   ｢FM-7/FM-NEW7/F
27A0: 4D 2D 37 37 20 CF BC DD BA DE 20 C6 AD B3 D3 DD :FE  M-77 ﾏｼﾝｺﾞ ﾆｭｳﾓﾝ
27B0: 20 CF C6 AD B1 D9 A3 20 C9 20 C0 D2 C9 20 CC DF :BE   ﾏﾆｭｱﾙ｣ ﾉ ﾀﾒﾉ ﾌﾟ
27C0: DB B8 DE D7 D1 20 C3 DE BD A1 4E 12 14 08 3C 3C :2C  ﾛｸﾞﾗﾑ ﾃﾞｽ｡N...<<
27D0: 20 B1 20 BF 20 CB DE 20 B6 20 C0 20 3E 3E 12 14 :F1   ｱ ｿ ﾋﾞ ｶ ﾀ >>..
27E0: 0A 20 B3 B4 B6 D7 20 B5 D8 C3 B8 D9 20 B4 B2 D8 :7D  . ｳｴｶﾗ ｵﾘﾃｸﾙ ｴｲﾘ
27F0: B1 DD 20 A6 20 B7 AC C9 DD 20 C9 20 CA B6 B2 20 :D8  ｱﾝ ｦ ｷｬﾉﾝ ﾉ ﾊｶｲ
2800: BA B3 BE DD 20 C3 DE 20 B9 DE B7 CA 20 BC C3 20 :C0  ｺｳｾﾝ ﾃﾞ ｹﾞｷﾊ ｼﾃ
2810: B8 C0 DE BB B2 A1 12 14 0C 38 3C 3C 20 B7 20 B0 :ED  ｸﾀﾞｻｲ｡...8<< ｷ ｰ
2820: 20 C9 20 BF 20 B3 20 BB 20 3E 3E 12 12 0D 20 20 :83   ﾉ ｿ ｳ ｻ >>...
2830: 20 20 98 95 99 20 20 20 20 20 20 20 20 20 20 20 :66    ┌─┐
2840: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 98 :78                 ┌
2850: 95 99 52 12 12 0E 3C 2D 2D 20 96 34 96 20 4C 45 :79  ─┐R...<-- |4| LE
2860: 46 54 20 4D 4F 56 45 20 20 2F 20 20 52 49 47 48 :CA  FT MOVE  /  RIGH
2870: 54 20 4D 4F 56 45 20 96 36 96 20 2D 2D 3E 12 12 :09  T MOVE |6| -->..
2880: 0F 20 20 20 20 9A 95 9B 20 20 20 20 20 20 20 20 :59  .    └─┘
2890: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 :00
28A0: 20 20 9A 95 9B 69 12 12 10 20 20 20 98 95 95 95 :5E    └─┘i...   ┌───
28B0: 95 95 99 20 20 20 20 20 20 20 20 20 98 95 95 95 :3A  ──┐         ┌───
28C0: 95 95 95 95 99 12 12 11 20 20 20 96 42 52 45 41 :32  ────┐...   |BREA
28D0: 4B 96 20 46 49 52 45 20 20 20 20 96 20 53 50 41 :41  K| FIRE    | SPA
28E0: 43 45 20 96 20 47 41 4D 45 20 53 54 41 52 54 20 :46  CE | GAME START
28F0: 12 12 12 20 20 20 9A 95 95 95 95 95 9B 20 20 20 :14  ...   └─────┘
2900: 20 20 20 20 20 20 9A 95 95 95 95 95 95 95 9B 35 :3D        └───────┘5
```

```
2910: 12 14 14 43 4F 50 59 52 49 47 48 54 20 28 43 29 :A7  ...COPYRIGHT (C)
2920: 20 31 39 38 34 20 62 79 20 48 2E 4E 41 4B 41 4D :EF   1984 by H.NAKAM
2930: 55 52 41 12 20 16 48 49 54 20 53 50 41 43 45 20 :C1  URA. .HIT SPACE
2940: 4B 45 59 20 21 2A 12 23 0F 47 41 4D 45 20 20 4F :41  KEY !*.#.GAME  O
2950: 56 45 52 12 1E 17 52 45 50 4C 41 59 20 3F 20 28 :A8  VER...REPLAY ? (
2960: 59 20 6F 72 20 4E 29 12 23 12 53 43 4F 52 45 3D :F1  Y or N).#.SCORE=
```

第＄F章

BASICとマシン語

——ＢＡＳＩＣとの連携——

1. ＵＳＲ文の実際

これまで作成したプログラムは、全てマシン語のみで書かれていました。つまりＢＡＳＩＣは全く使わずにプログラムを作っていたわけです。

しかし、大きなビジネスソフトなどでは、全てをマシン語で作成していたのでは、ソースプログラムが長くなりすぎて、作成の際やデバッグの際に非常に苦労することになります。そのためデバッグのしやすいＢＡＳＩＣの特性を活かして、メインプログラムをＢＡＳＩＣで作成し、処理時間のかかる部分のみをマシン語で作成し、それを共用して１つのプログラムを作成するということがよくあります。

ところが、ＢＡＳＩＣとマシン語を混用してプログラムを組む際に問題となるのは、ＢＡＳＩＣプログラムと、マシン語プログラムとの値の受け渡しです。ここまでの知識で考えられるのは、ＰＯＫＥ文で、メモリのどこかに値を格納して、マシン語の方はそれを参照して処理を行い、結果はまたメモリのどこかを経由して、ＰＥＥＫ文で受け取るという方法です。実際にはこの方法でも十分なのですが、ＰＯＫＥ文やＰＥＥＫ文では、１バイトの数しか読み書きできないので、整数（２バイトの長さがある）を値としてわたす場合は、アドレスをＡＤ、値をＶとすれば

```
POKE  AD, V  ¥  256
POKE  AD+1, V  MOD  256
```

というように、２つの文にしなければならず、非常に不便を強いられることになります。

そこで用いられるのが、ＵＳＲ文です。これはマシン語プログラムを１つの関数とみなして、

```
A=USR0 (B)
```

などとして用いるものです。ここではＢの値がマシン語プログラムへ渡され、マシン語プログラムから返された結果がＡに代入されるというわけです。この方法は、１つの値しかやりとりできないという欠点はあるものの、関数として用いることができるので便利です。そこでこの章ではＵＳＲ文の使い方を解説しましょう。

まず、USR文を使うときには使用するマシン語プログラムの実行開始番地を指定しなければなりません。それは

DEF　USR0=&H5000

などとします。USRのあとの数字は0から9までとることができるので，全部で10種類の関数を作成することができます。また数字は省略することもできて、省略すると0とみなされます。ですからこの例では、USR0を$5000番地からに指定しています。

それでは、関数の呼び出しはどうするかというと、他の関数と同じに、

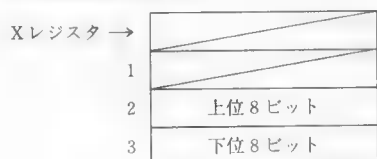
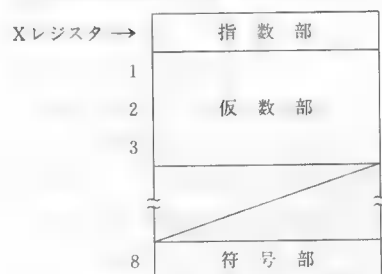
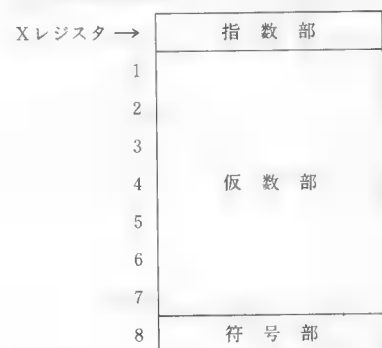
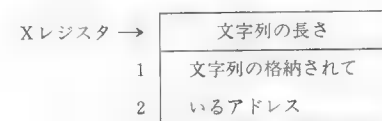
USR数字（引数）

として行います。ここでの引数は、文字式か数値式で、マシン語プログラムにはこれを計算した結果が渡されます。BASICの側では以上です。

次にマシン語プログラム側での使い方です。一番問題となるのは、引数の受渡しの方法です。

この受渡しで重要なのは、引数の型です。BASICには整数、単精度、倍精度、文字型の4種の型があります。ですから、形が違うと、引数の渡し方も異なってきます。そこでF−BASICでは、レジスタにその引数の情報を入れてマシン語サブルーチンを呼び出します。ですからマシン語サブルーチンでは、レジスタにセットされている情報から引数を受け取ることになります。

まずAレジスタには、引数の型を示す数字が格納されています。そしてXレジスタには引数の値が格納されているアドレスを示しています。その格納の形式を図F—1にあげました。単精度や倍精度実数の数値の表現方式などの詳しいことは解析マニュアルを参照してください。値を返すときにも、AレジスタとXレジスタに同じように値をセットして終了します。

① 整数型のとき　Aレジスタ=2

② 単精度実数型のとき　Aレジスタ=4

③ 倍精度実数型のとき　Aレジスタ=8

④ 文字型のとき　Aレジスタ=3

図F−1　USRでの値の受け取り方

2. ソートプログラムの作成

それではUSR文を作って、データのソートを行うプログラムを作成してみましょう。

簡単にするためにデータは整数型とし上昇順にソートすることにします。データのソートを行う場合、マシン語プログラムへ渡さなければならないのは、データの数(n)と、各要素のデータ(D_nとする）です。USR文では、1つの引数しか与えることはできないので、少し工夫しなければなりません。BASICには、変数の値の格納されている先頭番地を返す、VARPTRという関数があります。これと、配列とを組み合わせると、1つの引数でデータの数と各要素のデータをマシン語プログラムへ渡すことができます。

整数型の１次元配列は、図Ｆ―2のようにメモリ上に配置されています。そこでｎ個のデータを記憶するのに、Ｄ（１）～Ｄ（ｎ）に値を格納し、Ｄ（０）にデータの数を代入しておくと、データの数と各要素のデータを１つの配列に収めることができます。そして、マシン語プログラムへはこの配列の先頭番地、すなわち

VARPTR（Ｄ（０））

を与えてやるという方法です。このようにすればマシン語プログラムでは、与えられた配列の先頭番地から、すべてのデータを参照することができるというわけです。

次にアルゴリズムですが、ソートプログラムにはいろいろなアルゴリズムがあり、それぞれ処理方法、実行速度などが異なり、ソートプログラムだけで１冊の本を書くことができるほどです。ここでは、一番単純な方法として、単純選択ソートという方法を取ることにします。これは、データをD_iとするとき、D_1からD_nの中で最大なものを捜して、それをD_nと交換し、次に、D_1からD_{n-1}で最大なものを捜して、それをD_{n-1}と交換し……としていくソートの方法です。ＢＡＳＩＣで作成したものが図Ｆ―3です。ここでは、このプログラム中のサブルーチンの部分をマシン語にしてしまおうというわけです。

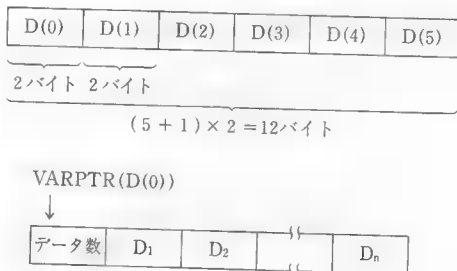
マシン語プログラムが図Ｆ―4です。プログラムの流れは図Ｆ―3のサブルーチンと同じです。ＦＯＲＪのループの中では、Ｄレジスタに、ＢＡＳＩＣでの変数M、Ｕレジスタに最大な数のあるアドレスを保持しています。またＹレジスタは、ＦＯＲＪのループのループ変数として用いています。

図Ｆ－２　配列の利用

〔図Ｆ－３　ＢＡＳＩＣによるソートプログラム〕

```
1000 ' ﾀﾝｼﾞｭﾝ ｿｰﾄ TEST PROGRAM
1010 DEFINT A-Z
1020 INPUT "NUMBER OF DATA =",N
1030 DIM D(N)
1040 D(0)=N
1050 FOR I=1 TO N
1060  D(I)=INT(RND*65536!-32768!)
1070  PRINT D(I),
1080 NEXT :PRINT
1090 PRINT "SORT START !!"
1100 TIME$="00:00:00"
1110 GOSUB 2000
1120 PRINT "SORT END    !!"
1130 T$=TIME$
1140 FOR I=1 TO N
1150  PRINT D(I),
1160 NEXT :PRINT
1170 PRINT "TIME=";T$
1180 END
1999 ' SORT PROGRAM ﾎﾝﾀｲ
2000 FOR I=N TO 2 STEP -1
2010   K=1:M=D(1)
2020   FOR J=2 TO I
2030     IF M<D(J) THEN K=J:M=D(J)
2040   NEXT J
2050   SWAP D(I),D(K)
2060 NEXT I
2070 RETURN
```

〔図Ｆ－５　ソートプログラムＢＡＳＩＣ部〕

```
1000 ' ﾀﾝｼﾞｭﾝ ｿｰﾄ TEST PROGRAM
1010 CLEAR ,&H5000
1020 DEFINT A-Z
1030 DEFUSR=&H5000
1040 LOADM "SORT*
1050 INPUT "NUMBER OF DATA =",N
1060 DIM D(N)
1070 D(0)=N
1080 FOR I=1 TO N
1090  D(I)=INT(RND*65536!-32768!)
1100  PRINT D(I),
1110 NEXT :PRINT
1120 PRINT "SORT START !!"
1130 TIME$="00:00:00"
1140 A=USR(VARPTR(D(0)))
1150 PRINT "SORT END    !!"
1160 T$=TIME$
1170 FOR I=1 TO N
1180  PRINT D(I),
1190 NEXT :PRINT
1200 PRINT "TIME=";T$
1210 END
```

そしてBASIC部が図F―5です。

以上のようにBASICプログラムの時間のかかる部分をマシン語に変換すると便利です。またこの方法では、BASICでプログラムのアルゴリズムの間違いをBASICで正してから、マシン語にすることもできて、ある意味では便利です（けれども、BASICの影響を受けたマシン語プログラムしかできないのも事実です）。

〔図F－4　マシン語によるソートプログラム〕

```
1                            * ﾀﾝｼﾞｭﾝ ｾﾝﾀｸ ｿｰﾄ
2    5000                          ORG    $5000
3                     5000  START  EQU    *
4
5    5000 34    76                 PSHS   D,X,Y,U ……レジスタを退避
6    5002 AE    02                 LDX    2,X ……D（0）のアドレスを得る
7    5004 EC    81                 LDD    ,X++ ……データの数nをロード：Xレジスタ←D（1）のアドレス
8    5006 27    28                 BEQ    END ……データがなければなにもせずに戻る
9    5008 20    21                 BRA    NEXTI ……NEXTIへ
10
11   500A 34    16          FORI   PSHS   D,X ……DレジスタとXレジスタを退避
12   500C 1F    02                 TFR    D,Y ……Yレジスタのカウンタをリセット
13   500E 33    84                 LEAU   ,X   ┐
14   5010 EC    81                 LDD    ,X++ ┘……最初のデータを最大と仮定する
15
16   5012 10A3  81          FORJ   CMPD   ,X++ ……これまでの最大値Mと比較
17   5015 2C    04                 BGE    NEXTJ ……もしM>=D（i）ならNEXTJへ
18   5017 33    1E                 LEAU   -2,X ┐
19   5019 EC    1E                 LDD    -2,X ┘……最大値を更新：Uレジスタ＝最大値のあるアドレス
20
21   501B 31    3F          NEXTJ  LEAY   -1,Y ……カウンタをデクリメント
22   501D 26    F3                 BNE    FORJ ……ループ
23
24   501F EC    C4                 LDD    ,U   ┐
25   5021 10AE  1E                 LDY    -2,X │
26   5024 ED    1E                 STD    -2,X │……最大値のデータと最後のデータとを交換
27   5026 10AF  C4                 STY    ,U   ┘
28   5029 35    16                 PULS   D,X ……DレジスタとXレジスタを復帰
29
30   502B 83    0001        NEXTI  SUBD   #1 ……FORIループのカウンタを1減らす
31   502E 26    DA                 BNE    FORI ……ループ
32   5030 35    F6          END    PULS   D,X,Y,U,PC ……終り

0 ERROR(S) DETECTED

SYMBOL TABLE:

END    5030   FORI   500A   FORJ   5012   NEXTI  502B   NEXTJ  501B
START  5000
```

〔図F－6　実行例〕

```
RUN
NUMBER OF DATA =10
 5968          -19138         3340           8903          -25946
 20075         -29957         29744         -9550           3702

SORT START !!
SORT END   !!
-29957         -25946        -19138         -9550           3340
 3702           5968          8903           20075          29744

TIME=00:00:00

Ready
```

3. 割込み

実践編もいよいよ最後の節です。ここでは（ＢＡＳＩＣとはあまり関係はないのですが）これまで何度か名前だけがでてきていた割込み（インタラプト）について解説します。

割込みというのは、ＣＰＵがある処理を実行している途中に、その処理を中断して別の処理を行い、その処理が終ったら、中断した箇所からまた仕事を始めるという動作のことをいいます。

ＢＡＳＩＣで用いられる

ON　KEY（n）　GOSUB　～

というステートメントがその割込みに関係した動作をするので、それを例に説明しましょう。図Ｆ—7のプログラムをみてください。このプログラムは、30行の“＊”の表示を繰り返すだけで、通常ならば決して50行は実行しません。しかし、ＰＦ１キーを押すと、プログラムの実行は50行に移り、“＋”を表示します。そして60行でその処理を終了してもとの“＊”の表示を繰り返します。

このように、ＣＰＵがある処理をしている状態で、任意のときに他の仕事を割込ますことができるわけです。この『任意のとき』というのは重要で、BSR などによるサブルーチンとの違いです。

ここで、ＰＦ１のキーを押すということを割込み要求、割込みによって、実行されるルーチンのことを割込み処理ルーチンといいます。

さて、話をマシン語の話に戻しましょう。マシン語には、ハードウェア割込みと呼ばれる、

ＲＥＳＥＴ，ＮＭＩ，ＦＩＲＱ，ＩＲＱ

の４つと、ソフトウェア割込みと呼ばれる

ＳＷＩ，ＳＷＩ２，ＳＷＩ３

の３つ、全部で７種類の割込みがあります。

ハードウェア割込みというのは、ハードウェアや人間による物理的な割込み要求によって生ずる割込みです。各割込みがどのような要求によって生ずるかを図Ｆ—8に示しました。

ソフトウェア割込みというのは、プログラム中に、 SWI などの命令を置くと、その命令を実行した際に割込みが生ずるものです。これは、ほとんどサブルーチンに近い動作をすることになります。

さて、ここでなにかの処理中に割込みが生じて割込み処理ルーチンの処理の後、元のルーチンに戻ってきたときのことを考えてください。割込みは（ハードウェア割込みの場合は特に）いつ生ずるかわかりませんから、割込み処理によってレジスタの内容がかわってしまったのでは、正常な処理は不可能です。これを避けるため，割込みが生じるとＣＰＵはその時点でのレジスタをＳスタックに退避します。すなわち自動的に、

PSHS CC, A, B, OP, X, Y, U, PC

を実行するわけです。そして、割込み処理ルーチンの最後は、 RTI 命令で終ります。この RTI 命令は、割込みが生じたとき、退避したレジスタを復帰し（少なくともレジスタに関しては）、何もなかったかのように、元のルーチンへ戻ります。

しかし、ＦＩＲＱとＲＥＳＥＴは例外です。まずＲＥＳＥＴは、ＣＰＵを完全に初期化するための割込みですから、レジスタの退避は行いません。そして、 RTI 命令で元のルーチンに戻るという

〔図Ｆ－７〕

```
10 ON KEY(1) GOSUB 50
20 KEY(1) ON
30 PRINT "*";
40 GOTO 30
50 PRINT "+";
60 RETURN
```

RESET	——	電源ON リセットスイッチ
NMI	——	FM－７では使っていない
FIRQ	——	BREAKキー サブCPUからの要求 （インターバルタイマ、クロック、ＰＦキーなど）
IRQ	——	タイマー割込みなど多数

図F－8　ハードウェア割込みの要因

ことはありません（この点からいうとＲＥＳＥＴは割込みと呼ばない方がよいかもしれません)。

ＦＩＲＱは、高速（Fast）割込みという意味で、割込みに反応する時間（レジスタを退避したりする時間）を短かくするために、レジスタはＣＣレジスタとＰＣ（プログラムカウンタ）しか退避しません。

次に割込みの優先順位というものを説明しましょう。もし、ＩＲＱとＦＩＲＱが同時に要求されたとすると、ＣＰＵはどちらの処理を行ったらよいかがわかりません。そこで、割込みには優先順位というものがあります。図Ｆ—9に示すように優先順位の高い割込み処理が取られることになっています。

また、いろいろな割込みに対して処理ルーチンが同じでは意味を持ちません。そこで各割込みについて、＄ＦＦＦ２～＄ＦＦＦＦ番地に割込みベクトルというものがあります。この割込みベクトルの割りあては図Ｆ—9のようになっています。例えば、ＦＩＲＱが生じると、ＣＰＵはレジスタを退避した後＄ＦＦＦ６、＄ＦＦＦ７に書かれている値をアドレスとする箇所にジャンプします。

さて、次には割込みに関係するＣＣレジスタ内のフラグについて解説しましょう。まずEフラグは、割込みが生じたときに、全レジスタを退避したか、しないか、すなわち、ＦＩＲＱであったかどうかを示すフラグです。ＣＣ，Ａ，Ｂ，……Ｙ，Ｕ，ＰＣのレジスタを退避したときには１、ＰＣとＣＣしか退避しなかったときは０になります。

F、Iフラグは、割込みをマスク（禁止）するためのフラグです。Ｆフラグが１のときにはＦＩＲＱ、Ｉフラグが１のときはＩＲＱの要求が生じても無視します。ですから割込みがかかってほしくないときには、これらのフラグをセットすればよいわけです。ＮＭＩとＲＥＳＥＴはマスクすることはできません。またＦ、Ｉフラグは割込み処理によって図Ｆ—9にフラグがセットされます。これにより，優先順位の高い割込みの処理中には、それより低い優先度の割込みは受け付けられません。

優先順位		割込みベクトル	変化するフラグ		
			E	F	I
高い	ＲＥＳＥＴ	＄ＦＦＦＥ～Ｆ		1	1
↑	ＮＭＩ	＄ＦＦＦＣ～Ｄ	1	1	1
	ＳＷＩ	＄ＦＦＦＡ～Ｂ	1	1	1
	ＦＩＲＱ	＄ＦＦＦ６～７	0	1	1
	ＩＲＱ	＄ＦＦＦ８～９	1		1
↓	ＳＷＩ２	＄ＦＦＦ４～５	1		
低い	ＳＷＩ３	＄ＦＦＦ２～３	1		

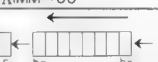
空白は変化しない。

図F－9　割込みの優先順位・他

以上、割込みについて述べましたが、これらの事項は相当高度なことですので、理解できなくても支障はないでしょう。

最後に、サブＣＰＵの制御において割込みを禁止した理由を述べておきます。既に示しましたがＢＲＥＡＫキーを押すとＦＩＲＱが生じます。通常の状態ではこのＦＩＲＱ割込みによってＢＡＳＩＣに対し、ＢＲＥＡＫキーが押されたことを通知して、ＢＡＳＩＣは適当な場所で処理を終るという具合になっています。また、ＢＲＥＡＫキーの処理ルーチンでは、サブＣＰＵに対しＢＲＥＡＫキーが押されたことを通知します。このサブＣＰＵへの通知が曲者で、(詳しくは述べませんが、＄Ｃ９５３からのルーチンを解析してみるとよいでしょう）この際、サブＣＰＵの停止を解除してしまいます。ですから、サブＣＰＵを停止している最中に、ＢＲＥＡＫキーによって、ＦＩＲＱが生じると、その結果サブＣＰＵの停止が解除されます。つまりコマンドなどのセットの途中で停止が解除されることになり、サブＣＰＵの暴走につながります。こういった理由から、サブＣＰＵを停止させている間は割込みを禁止しています（実際はＦＩＲＱだけマスクすればよいのですが、習慣でＩＲＱもマスクしています)。

実践編の最後に

さて、あなたはこれで、ＦＭ—7シリーズでのマシン語プログラミングに必要な最低限の事項をすべて習得することができたわけです。中には、よくわからずにとばしたところもあったかもしれません。全てを今すぐに理解する必要はないでしょうから、またいつか読み返すようにすればそれで十分です。

実践編では、プログラムのとらえ方からコーディングのしかた、そして、ゲームの作り方まで幅広く取りあげて解説してきました。しかし、これらは、少なからず受身の学習であったことと思います。ですから独自でプログラムを作成するとなると自信のない方もおられるでしょう。そういう方も含めて、これからは自分の独力で小さなプログラムでもいいですから作成してみて下さい。初めは、暴走したりでうまくいかないかもしれません。しかしこれを積み重ねれば、実力は飛躍的に伸びていきます。そして、少しずつ大きなプログラムへと移行していくとよいでしょう。

ある程度大きな（プリンタ用紙２枚以上の）プログラムを組むとなると、いわゆるテクニックを身に付ける必要ができるでしょう。そうなったら、他の人の作ったプログラム（やはりある程度長いもの）を解析することをお推めします。大物にチャレンジしたければ、Ｆ—ＢＡＳＩＣなどはいい教材となります（解析マニュアルを参照しながら行えば、非常に多くのテクニックが習得できるでしょう)。さらに実践的なプログラムに興味のある人は、弊社の『ユーティリティプログラム応用実例集』のプログラムなどが最適です。各プログラムの構造が詳しく解説されています。このように既に作られたプログラムを解析するというのは、意外と上達の早道ともいえます。

いずれにしてもこれでFM—7シリーズでのマシン語の基礎は習得できました。この基礎を活かして、マシン語プログラミングの腕をみがいてください。

付　録

図付録—I

インストラクション	ニーモニック	アドレッシングモード															動作	5	3	2	1	0
		イミディエイト			ダイレクト			インデックス[1]			エクステンド			インヘレント								
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#		H	N	Z	V	C
ABX														3A	3	1	B+X→X(Unsigned)	●	●	●	●	●
ADC	ADCA	89	2	2	99	4	2	A9	4+	2+	B9	5	3				A+M+C→A	↕	↕	↕	↕	↕
	ADCB	C9	2	2	D9	4	2	E9	4+	2+	F9	5	3				B+M+C→B	↕	↕	↕	↕	↕
ADD	ADDA	8B	2	2	9B	4	2	AB	4+	2+	BB	5	3				A+M→A	↕	↕	↕	↕	↕
	ADDB	CB	2	2	DB	4	2	EB	4+	2+	FB	5	3				B+M→B	↕	↕	↕	↕	↕
	ADDD	C3	4	3	D3	6	2	E3	6+	2+	F3	7	3				D+M:M+1→D	●	↕	↕	↕	↕
AND	ANDA	84	2	2	94	4	2	A4	4+	2+	B4	5	3				A∧M→A	●	↕	↕	0	●
	ANDB	C4	2	2	D4	4	2	E4	4+	2+	F4	5	3				B∧M→B	●	↕	↕	0	●
	ANDCC	1C	3	2													CC∧IMM→CC					6
ASL	ASLA													48	2	1		7	↕	↕	↕	↕
	ASLB													58	2	1		7	↕	↕	↕	↕
	ASL				08	6	2	68	6+	2+	78	7	3					7	↕	↕	↕	↕
ASR	ASRA													47	2	1		7	↕	↕	●	↕
	ASRB													57	2	1		7	↕	↕	●	↕
	ASR				07	6	2	67	6+	2+	77	7	3					7	↕	↕	●	↕
BIT	BITA	85	2	2	95	4	2	A5	4+	2+	B5	5	3			1	Bit Test A(M∧A)	●	↕	↕	0	●
	BITB	C5	2	2	D5	4	2	E5	4+	2+	F5	5	3			1	Bit Test B(M∧B)	●	↕	↕	0	●
CLR	CLRA													4F	2	1	0→A	●	0	1	0	0
	CLRB													5F	2	1	0→B	●	0	1	0	0
	CLR				0F	6	2	6F	6+	2+	7F	7	3				0→M	●	0	1	0	0
CMP	CMPA	81	2	2	91	4	2	A1	4+	2+	B1	5	3				Compare M from A	7	↕	↕	↕	↕
	CMPB	C1	2	2	D1	4	2	E1	4+	2+	F1	5	3				Compare M from B	7	↕	↕	↕	↕
	CMPD	10	5	4	10	7	3	10	7+	3+	10	8	4				Compare M:M+1 from D	●	↕	↕	↕	↕
		83			93			A3			B3											
	CMPS	11	5	4	11	7	3	11	7+	3+	11	8	4				Compare M:M+1 from S	●	↕	↕	↕	↕
		8C			9C			AC			BC											
	CMPU	11	5	4	11	7	3	11	7+	3+	11	8	4				Compare M:M+1 from U	●	↕	↕	↕	↕
		83			93			A3			B3											
	CMPX	8C	4	3	9C	6	2	AC	6+	2+	BC	7	3				Compare M:M+1 from X	●	↕	↕	↕	↕
	CMPY	10	5	4	10	7	3	10	7+	3+	10	8	4				Compare M:M+1 from Y	●	↕	↕	↕	↕
		8C			9C			AC			BC											
COM	COMA													43	2	1	$\overline{A}$→A	●	↕	↕	0	1
	COMB													53	2	1	$\overline{B}$→B	●	↕	↕	0	1
	COM				03	6	2	63	6+	2+	73	7	3				$\overline{M}$→M	●	↕	↕	0	1
CWAI		3C	≥20	2													CC∧IMM→CC Wait for Interrupt					6
DAA														19	2	1	Decimal Adjust A	●	↕	↕	7	↕
DEC	DECA													4A	2	1	A−1→A	●	↕	↕	↕	●
	DECB													5A	2	1	B−1→B	●	↕	↕	↕	●
	DEC				0A	6	2	6A	6+	2+	7A	7	3				M−1→M	●	↕	↕	↕	●
EOR	EORA	88	2	2	98	4	2	A8	4+	2+	B8	5	3				A∀M→A	●	↕	↕	0	●
	EORB	C8	2	2	D8	4	2	E8	4+	2+	F8	5	3				B∀M→B	●	↕	↕	0	●
EXG	R1, R2	1E	8	2													R1↔R2[2]	●	●	●	●	●
INC	INCA													4C	2	1	A+1→A	●	↕	↕	↕	●
	INCB													5C	2	1	B+1→B	●	↕	↕	↕	●
	INC				0C	6	2	6C	6+	2+	7C	7	3				M+1→M	●	↕	↕	↕	●
JMP					0E	3	2	6E	3+	2+	7E	4	3				EA[3]→PC	●	●	●	●	●
JSR					9D	7	2	AD	7+	2+	BD	8	3				Jump to Subroutine	●	●	●	●	●
LD	LDA	86	2	2	96	4	2	A6	4+	2+	B6	5	3				M→A	●	↕	↕	0	●
	LDB	C6	2	2	D6	4	2	E6	4+	2+	F6	5	3				M→B	●	↕	↕	0	●
	LDD	CC	3	3	DC	5	2	EC	5+	2+	FC	6	3				M:M+1→D	●	↕	↕	0	●
	LDS	10	4	4	10	6	3	10	6+	3+	10	7	4				M:M+1→S	●	↕	↕	0	●
		CE			DE			EE			FE											
	LDU	CE	3	3	DE	5	2	EE	5+	2+	FE	6	3				M:M+1→U	●	↕	↕	0	●
	LDX	8E	3	3	9E	5	2	AE	5+	2+	BE	6	3				M:M+1→X	●	↕	↕	0	●
	LDY	10	4	4	10	6	3	10	6+	3+	10	7	4				M:M+1→Y	●	↕	↕	0	●
		8E			9E			AE			BE											
LEA	LEAS							32	4+	2+							EA[3]→S	●	●	●	●	●
	LEAU							33	4+	2+							EA[3]→U	●	●	●	●	●
	LEAX							30	4+	2+							EA[3]→X	●	●	↕	●	●
	LEAY							31	4+	2+							EA[3]→Y	●	●	↕	●	●
LSL	LSLA													48	2	1		7	↕	↕	↕	↕
	LSLB													58	2	1		7	↕	↕	↕	↕
	LSL				08	6	2	68	6+	2+	78	7	3					7	↕	↕	↕	↕
LSR	LSRA													44	2	1		●	0	↕	●	↕
	LSRB													54	2	1		●	0	↕	●	↕
	LSR				04	6	2	64	6+	2+	74	7	3					●	0	↕	●	↕
MUL														3D	11	1	A×B→D (Unsigned)	●	●	↕	●	■

インストラクション	ニーモニック	アドレッシングモード															動作	5	3	2	1	0
		イミディエイト			ダイレクト			インデックス[1]			エクステンド			インヘレント				H	N	Z	V	C
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#						
NEG	NEGA													40	2	1	$\overline{A}+1\rightarrow A$	7	↕	↕	↕	↕
	NEGB													50	2	1	$\overline{B}+1\rightarrow B$	7	↕	↕	↕	↕
	NEG				00	6	2	60	6+	2+	70	7	3				$\overline{M}+1\rightarrow M$	7	↕	↕	↕	↕
NOP														12	2	1	No Operation	●	●	●	●	●
OR	ORA	8A	2	2	9A	4	2	AA	4+	2+	BA	5	3				A V M→A	●	↕	↕	0	●
	ORB	CA	2	2	DA	4	2	EA	4+	2+	FA	5	3				B V M→B	●	↕	↕	0	●
	ORCC	1A	3	2													CC V IMM→CC				6	
PSH	PSHS	34	5+[4]	2													Push Registers on S Stack	●	●	●	●	●
	PSHU	36	5+[4]	2													Push Registers on U Stack	●	●	●	●	●
PUL	PULS	35	5+[4]	2													Pull Registers from S Stack	●	●	●	●	●
	PULU	37	5+[4]	2													Pull Registers from U Stack	●	●	●	●	●
ROL	ROLA													49	2	1	A B M: c ← b7 … b0 ←	●	↕	↕	↕	↕
	ROLB													59	2	1		●	↕	↕	↕	↕
	ROL				09	6	2	69	6+	2+	79	7	3					●	↕	↕	↕	↕
ROR	RORA													46	2	1	A B M: → c → b7 … b0	●	↕	↕	●	↕
	RORB													56	2	1		●	↕	↕	●	↕
	ROR				06	6	2	66	6+	2+	76	7	3					●	↕	↕	●	↕
RTI														3B	6/15	1	Return from Interrupt					6
RTS														39	5	1	Return from Subroutine	●	●	●	●	●
SBC	SBCA	82	2	2	92	4	2	A2	4+	2+	B2	5	3				A-M-C→A	7	↕	↕	↕	↕
	SBCB	C2	2	2	D2	4	2	E2	4+	2+	F2	5	3				B-M-C→B	7	↕	↕	↕	↕
SEX														1D	2	1	Sign Extend B into A	●	↕	↕	0	●
ST	STA				97	4	2	A7	4+	2+	B7	5	3				A→M	●	↕	↕	0	●
	STB				D7	4	2	E7	4+	2+	F7	5	3				B→M	●	↕	↕	0	●
	STD				DD	5	2	ED	5+	2+	FD	6	3				D→M:M+1	●	↕	↕	0	●
	STS				10	6	3	10	6+	3+	10	7	4				S→M:M+1	●	↕	↕	0	●
					DF			EF			FF											
	STU				DF	5	2	EF	5+	2+	FF	6	3				U→M:M+1	●	↕	↕	0	●
	STX				9F	5	2	AF	5+	2+	BF	6	3				X→M:M+1	●	↕	↕	0	●
	STY				10	6	3	10			10	7	4				Y→M:M+1	●	↕	↕	0	●
					9F			AF	6+	3+	BF											
SUB	SUBA	80	2	2	90	4	2	A0	4+	2+	B0	5	3				A-M→A	7	↕	↕	↕	↕
	SUBB	C0	2	2	D0	4	2	E0	4+	2+	F0	5	3				B-M→B	7	↕	↕	↕	↕
	SUBD	83	4	3	93	6	2	A3	6+	2+	B3	7	3				D-M:M+1→D	●	↕	↕	↕	↕
SWI	SWI[5]													3F	19	1	Software Interrupt 1	●	●	●	●	●
	SWI2[5]													10	20	·2	Software Interrupt 2	●	●	●	●	●
														3F								
	SWI3[5]													11	20	1	Software Interrupt 3	●	●	●	●	●
														3F								
SYNC														13	≧4	1	Synchronize to Interrupt	●	●	●	●	●
TFR	R1, R2													1F	6	2	R1→R2[2]	●	●	●	●	●
TST	TSTA													4D	2	1	Test A	●	↕	↕	0	●
	TSTB													5D	2	1	Test B	●	↕	↕	0	●
	TST				0D	6	2	6D	6+	2+	7D	7	3				Test M	●	↕	↕	0	●

Op　オペレーションコード(16進数)
~　MPUの実行サイクル数
#　命令語のバイト数
+　算術加算
　算術減算
●　算術乗算

$\overline{M}$　Mの補数
→　転送方向
↕　命令実行の結果に応じて変化
●　その命令では変化しない
CC　コンディションコードレジスタ
:　比較
V　論理和
Λ　論理積
⊻　排他的論理和

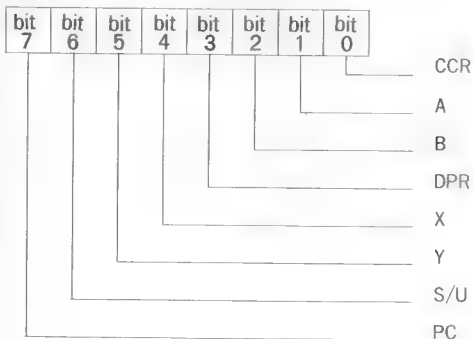
E　エンタイアフラグ
F　FIRQマスクフラグ
H(b_5)　b_3からのハーフキャリー
I　IRQマスクフラグ
N(b_3)　負のフラグ
Z(b_2)　ゼロのフラグ
V(b_1)　2の補数のオーバフローフラグ
C(b_0)　b_7からのキャリー　b_7へのボロー

インストラクション	ニーモニック	Op	～ #	#	動作	5 H	3 N	2 Z	1 V	0 C
BCC	BCC	24	3	2	Branch C=0	●	●	●	●	●
	LBCC	10 24	5(6)	4	Long Branch C=0	●	●	●	●	●
BCS	BCS	25	3	2	Branch C=1	●	●	●	●	●
	LBCS	10 25	5(6)	4	Long Branch C=1	●	●	●	●	●
BEQ	BEQ	27	3	2	Branch Z=1	●	●	●	●	●
	LBEQ	10 27	5(6)	4	Long Branch Z=1	●	●	●	●	●
BGE	BGE	2C	3	2	Branch≧Zero	●	●	●	●	●
	LBGE	10 2C	5(6)	4	Long Branch≧Zero	●	●	●	●	●
BGT	BGT	2E	3	2	Branch>Zero	●	●	●	●	●
	LBGT	10 2E	5(6)	4	Long Branch>Zero	●	●	●	●	●
BHI	BHI	22	3	2	Branch Higher	●	●	●	●	●
	LBHI	10 22	5(6)	4	Long Branch Higher	●	●	●	●	●
BHS	BHS	24	3	2	Branch Higher or Same	●	●	●	●	●
	LBHS	10 24	5(6)	4	Long Branch Higher or Same	●	●	●	●	●
BLE	BLE	2F	3	2	Branch≦Zero	●	●	●	●	●
	LBLE	10 2F	5(6)	4	Long Branch≦Zero	●	●	●	●	●
BLO	BLO	25	3	2	Branch lower	●	●	●	●	●
	LBLO	10 25	5(6)	4	Long Branch Lower	●	●	●	●	●

インストラクション	ニーモニック	アドレッシングモード リラティブ Op	～	##	動作	5 H	3 N	2 Z	1 V	0 C
BLS	BLS	23	3	2	Branch Lower or Same	●	●	●	●	●
	LBLS	10 23	5(6)	4	Long Branch Lower or Same	●	●	●	●	●
BLT	BLT	2D	3	2	Branch<Zero	●	●	●	●	●
	LBLT	10 2D	5(6)	4	Long Branch<Zero	●	●	●	●	●
BMI	BMI	2B	3	2	Branch Minus	●	●	●	●	●
	LBMI	10 2B	5(6)	4	Long Branch Minus	●	●	●	●	●
BNE	BNE	26	3	2	Branch Z=0	●	●	●	●	●
	LBNE	10 26	5(6)	4	Long Branch Z=0	●	●	●	●	●
BPL	BPL	2A	2	2	Branch Plus	●	●	●	●	●
	LBPL	10 2A	5(6)	4	Long Branch Plus	●	●	●	●	●
BRA	BRA	20	3	2	Branch Ahways	●	●	●	●	●
	LBRA	16	5	4	Long Branch Always	●	●	●	●	●
BRN	BRN	21	3	2	Branch Never	●	●	●	●	●
	LBRN	10 21	5	4	Long Branch Never	●	●	●	●	●
BSR	BSR	8D	7	2	Branch to Subroutine	●	●	●	●	●
	LBSR	17	9	3	Long Branch to Subroutine	●	●	●	●	●
BVC	BVC	28	3	2	Branch V=0	●	●	●	●	●
	LBVC	10 28	5(6)	4	Long Branch V=0	●	●	●	●	●
BVS	BVS	29	3	2	Branch V=1	●	●	●	●	●
	LBVS	10 29	5(6)	4	Long Branch V=1	●	●	●	●	●

ブランチ命令表

注：
1．インデックスアドレッシングモードの時は，この欄に書かれているバイト数，実行サイクル数に，「インデックスアドレッシングモードのポストバイト」の表にある値を加えたものが実際の値となります。
2．R 1とR 2は，同じバイト数のレジスタでなければいけません。
　8ビットレジスタ：A，B，CC，DP
　16ビットレジスタ：X，Y，U，S，D，PC
3．EAは，実効アドレス（Effective address）を示します。
4．PSH，PUL命令は，この値に，プッシュ又はプルするバイト数を加えた値となります。
5．SWIはIフラグとFフラグを1にしますが，SWI 2，SWI 3は，Iフラグ，Fフラグには影響しません。
6．CCレジスタは，命令の結果によってセット又はリセットされます。
7．未定義
8．MUL命令の時に限り，Cフラグはdit 7と同じ値をとります。

PUSH/PULLのポストバイト

TRANSFER/EXCHANGEのポストバイト

SOURCE	DESTINATION

0000＝D（A：B）	0101＝PC
0001＝X	1000＝A
0010＝Y	1001＝B
0011＝U	1010＝CCR
0100＝S	1011＝DPR

アドレッシング モード			インダイレクトでない場合				インダイレクトの場合			
			アセンブラ形式	ポスト バイト	+~	+#	アセンブラ形式	ポスト バイト	+~	+#
インデックスト	コンスタントオフセット	オフセットなし	,R	1RR00100	0	0	[,R]	1RR10100	3	0
		5ビット オフセット	n,R	0RRnnnnn	1	0	,―	―	―	―
		8ビット オフセット	n,R	1RR01000	1	1	[n,R]	1RR11000	4	1
		16ビット オフセット	n,R	1RR01001	4	2	[n,R]	1RR11001	7	2
	アキュムレータオフセット	Aレジスタ オフセット	A,R	1RR00110	1	0	[A,R]	1RR10110	4	0
		Bレジスタ オフセット	B,R	1RR00101	1	0	[B,R]	1RR10101	4	0
		Dレジスタ オフセット	D R	1RR01011	4	0	[D,R]	1RR11011	7	0
	オートインクリメント/デクリメント	インクリメント(+1)	,R+	1RR00000	2	0	―	―	―	―
		インクリメント(+2)	,R++	1RR00001	3	0	[,R++]	1RR10001	6	0
		デクリメント (−1)	,−R	1RR00010	2	0	―	―	―	―
		デクリメント (−2)	,−−R	1RR00011	3	0	[,−−R]	1RR10011	6	0
プログラムカウンタリラティブ		8ビット オフセット	n,PCR	1XX01100	1	1	[n,PCR]	1XX11100	4	1
		16ビット オフセット	n,PCR	1XX01101	5	2	[n,PCR]	1XX11101	8	2
エクステンディットインダイレクト		16ビット アドレス	―	―	―	―	[n]	10011111	5	2

インデックスアドレッシングモードのポストバイト

〈記号〉 R：X R：00＝X X：Don't care ~＝追加されるマシンサイクル数
Y 01＝Y #＝追加されるバイト数
U 10＝U
S 11＝S

ブランチ	OP	~	#
BRA	20	3	2
LBRA	16	5	3
BRN	21	3	2
LBRN	1021	5	4
BSR	8D	7	2
LBSR	17	9	3

符号付きブランチ

Test	True	OP	False	OP
r>m	BGT	2E	BLE	2F
r≧m	BGE	2C	BLT	2D
r=m	BEQ	27	BNE	26
r≦m	BLE	2F	BGT	2E
r<m	BLT	2D	BGE	2C

条件付きブランチ

Test	True	OP	False	OP
N=1	BMI	2B	BPL	2A
Z=1	BEQ	27	BNE	26
V=1	BVS	29	BVC	28
C=1	BCS	25	BCC	24

符号なしブランチ

Test	True	OP	False	OP
r>m	BHI	22	BLS	23
r≧m	BHS	24	BLO	25
r=m	BEQ	27	BNE	26
r≦m	BLS	23	BHI	22
r<m	BLO	25	BHS	24

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	ダイレクト		リラティブ		A	B	インデックス	エクステンド	イミディエイト	ダイレクト	インデックス	エクステンド	イミディエイト	ダイレクト	インデックス	エクステンド
0	NEG	PAGE2	BRA	LEAX	NEG				SUBA				SUBB			
1	—	PAGE3	BRN/LBRN	LEAY	—				CMPA				CMPB			
2	—	NOP	BHI/LBHI	LEAS	—				SBCA				SBCB			
3	COM	SYNC	BLS/LBLS	LEAU	COM				SUBD/CMPD/CMPU				ADDD			
4	LSR	—	BHS/* LBLO	PSHS	LSR				ANDA				ANDB			
5	—	—	** BLO/LBLO	PULS	—				BITA				BITB			
6	ROR	LBRA	BNE/LBNE	PSHU	ROR				LDA				LDB			
7	ASR	LBSR	BEQ/LBEQ	PULU	ASR				—	STA			—	STB		
8	ASL (LSL)	—	BVC/LBVC	—	ASL (LSL)				EORA				EORB			
9	ROL	DAA	BVS/LBVS	RTS	ROL				ADCA				ADCB			
A	DEC	ORCC	BPL/LBPL	ABX	DEC				ORA				ORB			
B	—	—	BMI/LBMI	RTI	—				ADDA				ADDB			
C	INC	ANDCC	BGE/LBGE	CWAI	INC				CMPX/CMPY/CMPS				LDD			
D	TST	SEX	BLT/LBLT	MUL	TST				BSR	JSR			—	STD		
E	JMP	EXG	BGT/LBGT	—	—		JMP		LDX/LDY				LDU/LDS			
F	CLR	TFR	BLE/LBLE	SWI/2/3	CLR				—	STX/STY			—	STU/STS		

* BCC/LBCC　* * BCS/LBCS

図付録－2　逆アセンブル表

hex	$x000	$0x00	$00x0	$000x	hex
1	4096	256	16	1	1
2	8192	512	32	2	2
3	12288	768	48	3	3
4	16384	1024	64	4	4
5	20480	1280	80	5	5
6	24576	1536	96	6	6
7	28672	1792	112	7	7
8	32768	2048	128	8	8
9	36864	2304	144	9	9
A	40960	2560	160	10	A
B	45056	2816	176	11	B
C	49152	3072	192	12	C
D	53248	3328	208	13	D
E	57344	3584	224	14	E
F	61440	3840	240	15	F

図付録－3　16進変換表(4ケタ)

2　進　数	16進数	10進数
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

図付録－4　2↔16↔10進変換

上位4ビット→　下位4ビット↓

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DE		0	@	P	`	p				ー	タ	ミ		
1	SH	D1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	SX	D2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	EX	D3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	ET	D4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	EQ	NK	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	AK	SN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	BL	EB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	BS	CN	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	〒
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	市
A	LF	SB	*	:	J	Z	j	z			ェ	コ	ハ	レ	♦	区
B	HM	EC	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	町
C	CL	→	,	<	L	¥	l	¦			ャ	シ	フ	ワ	●	村
D	CR	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	人
E	SO	↑	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	SI	↓	/	?	O	_	o	DL			ッ	ソ	マ	゜		

図付録－5　キャラクタコード表

リクエスト名	RCB＋0	＋1	＋2	＋3	＋4	＋5	＋6	＋7	備考
ANALGP	$00	$00	チャンネル 0～3	電圧レンジ 'H'or'L'	A/D変換データ				アナログポートからA/D変換されたデータを読み取る
MOTOR	$01	$00	モータクラブ ON=$FF						オーディオカセットのモータをコントロールする
CTBWRT	$02	$00	書込みデータ						カセットテープに1バイトのデータを書き込む
CTBRED	$03	$00	読み込んだデータ						カセットテープから1バイト読み込む
SCREEN	$05	$00	ワークエリア(209バイト)先頭アドレス		黒色指定				CRT画面をプリンタにコピー（HARDC 2）
RESTOR	$08	エラー番号						ドライブ 0～3	ディスクをリストアする
DWRITE	$09	エラー番号	出力データ先頭アドレス		トラック 0～39	セクタ 1～16	サイド 0，1	ドライブ 0～3	ディスクへ1セクタ書き込む
DREAD	$0A	エラー番号	入力データバッファ先頭アドレス		トラック 0～39	セクタ 1～16	サイド 0，1	ドライブ 0～3	ディスクから1セクタ読み込む
BEEPON BEEPOF	$0C $0D	$00							ブザー音をコントロールする
LPOUT	$0E	$00	出力データ先頭アドレス		出力データバイト数				プリンタへ文字列を出力
HDCOPY	$0F	$00	ワークエリア(209バイト）先頭アドレス		黒色指定	灰色指定			CRT画面をプリンタにコピー（HARDC 1）
SUBOUT	$10	エラー番号	転送データ先頭アドレス		転送データ長（1～128)				サブシステムへコマンドを渡します
SUBIN	$11	エラー番号	データバッファ先頭アドレス		出力データ長（1～128)		データバッファ長（0～128)		サブシステムへコマンドを渡し結果を受けとります
INPUT INPUTC	$12 $13	エラー番号	データ領域先頭アドレス		出力データ長		入力データバッファ長		サブシステムにより1行入力を行います　(INPUTCは次を入力)
OUTPUT	$14	エラー番号	出力データ先頭アドレス		出力データ長				サブシステムにより文字列出力を行います
KEYIN	$15	エラー番号	キー入力データバッファ先頭アドレス		データバッファ長（＝2）				サブシステムより、キーボードの情報を受けとる
KANJIR	$16	$00	データバッファ先頭アドレス		JISコード				漢字ROMから漢字のドットパターンを得る
LPCHK	$17	エラー番号							プリンタのチェック
BIINIT	$18	$00							BIOSのイニシャライズを行う

注）◤は復帰情報

図付録－6　BIOSリクエスト一覧（バブル関係を除く）

◀メインCPU▶
$0000
$4000
$8000
$C000
$FC00
$FFFF
RAM
BASIC
ROM
RAMモード
ROMモード
$FC00
$FC80
$FD00
$FE00
$FF00
$FFE0
$FFF0
$FFFF
RAM
(128B)
共有RAM
(128B)
サブCPU
I/O
領域
ブートROM
RAM
$FFF0
$FFF2
$FFF4
$FFF6
$FFF8
$FFFA
$FFFC
$FFFE
SWI3ベクトル
SWI2ベクトル
FIRQベクトル
IRQベクトル
SWIベクトル
NMIベクトル
ROM RESETベクトル
◀サブCPU▶
$0000
$4000
$8000
$C000
$D000
$E000
$FFFF
VRAM
(B)
VRAM
(R)
VRAM
(G)
コンソール
バッファ
キャラクタROM
ROM
(CRTモニタ)
$D000
$D380
$D400
$D7FF
ワーク
RAM
(896B)
共有RAM
メインCPU
I/O
領域

図付録−7 メモリマップ

〔図付録—8〕逆アセンブラ〕

```
100 ' DISASMB v1.0
110 DEFFNA$(X)=RIGHT$("000"+HEX$(X),4)
120 DEFFND$(X)=RIGHT$("0"+HEX$(X),2)
130 GOSUB 3000 :' data read
140 PRINT "***  FM-7 dis asmb V1.0  ***"
150 INPUT "start address =$",S$:BEGIN=VAL("&H"+S$)
160 INPUT "end   address =$",S$
170 IF S$="" THEN DEND=BEGIN+20 ELSE DEND=VAL("&H"+S$)
180 PRINT
500 AD=BEGIN:WHILE AD<=DEND
510 SAD=AD:GOSUB 1000
520 PRINT FNA$(SAD);"-";
530 FOR I=SAD TO AD-1:PRINT FND$(PEEK(I));" ";:NEXT
540 PRINT TAB(24);OPC$;TAB(32);OPR$
550 WEND
560 A$=INPUT$(1)
570 IF A$=" " THEN BEGIN=AD:DEND=BEGIN+20:GOTO 500
580 IF A$=CHR$(13) THEN 150
590 END
1000 ' 1 line dis asmb
1010 PF=0
1020 OP=PEEK(AD):AD=AD+1
1030 IF OP=&H10 OR OP=&H11 THEN IF PF<>0 THEN 1490 ELSE PF=OP:GOTO 1020
1040 MSN=OP ¥16 :LSN=OP MOD 16
1050 ON MSN+1 GOTO 1070,1080,1140,1150
1060 ON MSN¥4 GOTO 1280,1330,1430:GOTO 1490
1070 OPC$=NO$(LSN):IF OPC$="?" THEN 1490 ELSE 2010:' ♠♠♠♠ MSN=0
1080 OPC$=N1$(LSN):IF OPC$="?" THEN 1490 :' ♠♠♠♠ MSN=1
1090 IF LSN=2 OR LSN=3 OR LSN=9 OR LSN=&HD THEN 2080
1100 IF LSN=6 OR LSN=7 THEN 2060
1110 IF LSN=&HA OR LSN=&HC THEN 2090
1120 D=PEEK(AD):AD=AD+1
1130 OPR$=REG$(D¥16)+","+REG$(D MOD16):RETURN
1140 OPC$=N2$(LSN):IF PF=&H11 THEN OPC$="L"+OPC$:GOTO 2060 ELSE 2040:'♠♠♠ MSN=2
1150 OPC$=N3$(LSN):'♠♠♠ MSN=3
1160 IF OPC$="?" THEN 1490
1170 IF 0=<LSN AND LSN<=3 THEN 2120
1180 IF LSN<>&HF THEN 1200
1190 IF PF=0 THEN 2080 ELSE IF PF=&H10 THEN OPC$="SWI2":GOTO 2080 ELSE OPC$="SWI
3":GOTO 2080
1200 IF LSN=&HC THEN 2090
1210 IF LSN>7 THEN 2080
1220 D=PEEK(AD):AD=AD+1:OPR$="":IF D=0 THEN 1270
1230 FOR I=0TO7:B=(D¥(2^I))MOD2
1240 IF I=6 AND B=1 THEN IF LSN=4 OR LSN=5 THEN OPR$=OPR$+"U,":GOTO 1260 : ELSE
OPR$=OPR$+"S,":GOTO 1260
1250 IF B=1 THEN OPR$=OPR$+PREG$(I)+","
1260 NEXT :OPR$=LEFT$(OPR$,LEN(OPR$)-1)
1270 RETURN
1280 OPC$=NO$(LSN):' ♠♠♠ MSN=4..7
1290 IF OPC$="?" THEN 1490
1300 IF OP=&H4E OR OP=&H5E THEN 1490
1310 ON MSN-3 GOTO 2020,2030,2120,2110
1320 GOTO 1490
1330 OPC$=N8$(LSN):IF OP=&H87 OR OP=&H8F THEN 1490 :' ♠♠♠ MSN=8..B
1340 IF OPC$="?" THEN 1490
1350 IF OP=&H8D THEN OPC$="BSR":GOTO 2040
1360 IF PF=0 THEN 1410
1370 IF LSN=3 THEN IF PF=&H10 THEN OPC$="CMPD" ELSE OPC$="CMPU"
1380 IF LSN=&HC THEN IF PF=&H10 THEN OPC$="CMPY" ELSE OPC$="CMPS"
1390 IF LSN=&HE THEN IF PF=&H10 THEN OPC$="LDY" ELSE 1490
1400 IF LSN=&HF THEN IF PF=&H10 THEN OPC$="STY" ELSE 1490
1410 IF MSN=8 THEN IF LSN=3 OR LSN>=&HC THEN 2100 ELSE 2090
1420 ON MSN-8 GOTO 2010,2120,2110:GOTO 1490
1430 OPC$=NC$(LSN):IF OP=&HC7 OR OP=&HCD OR OP=&HCF THEN 1490:'♠♠♠ MSN=C..F
1440 IF OPC$="?" THEN 1490
1450 IF PF=0 THEN 1480
1460 IF LSN=&HE THEN IF PF=&H10 THEN OPC$="LDS" ELSE 1490
1470 IF LSN=&HF THEN IF PF=&H10 THEN OPC$="STS" ELSE 1490
1480 MSN=MSN-4:GOTO 1410
1490 OPC$="?":OPR$="":AD=SAD+1:RETURN:' ERROR
2000 '** addressing mode
2010 OPR$="$"+FND$(PEEK(AD)):AD=AD+1:RETURN:' DIRECT
2020 OPC$=OPC$+"A":OPR$="":RETURN:' INHERENT Acc A
2030 OPC$=OPC$+"B":OPR$="":RETURN:' INHERENT Acc B
2040 R=PEEK(AD):AD=AD+1:IF R>127 THEN R=R-256:' RELATIVE #1
2050 A=AD+R:OPR$="$"+FNA$(A):RETURN
2060 R=PEEK(AD)*256+PEEK(AD+1):AD=AD+2:IF R>32767 THEN R=R-65536!:' RELATIVE #2
2070 A=AD+R:OPR$="$"+FNA$(A):RETURN
2080 OPR$="":RETURN:' INHERENT
2090 OPR$="#$"+FND$(PEEK(AD)):AD=AD+1:RETURN:' IMMED #1
2100 OPR$="#$"+FNA$(PEEK(AD)*256+PEEK(AD+1)):AD=AD+2:RETURN:' IMMED #2
2110 OPR$="$"+FNA$(PEEK(AD)*256+PEEK(AD+1)):AD=AD+2:RETURN:' EXTENDED
2120 ' INDEX
2130 P=PEEK(AD):AD=AD+1
2140 R$=IREG$((P AND &H60)¥32)
```

```
2150 IF (P AND &H80)=0 THEN 2330
2160 IF P AND &H10 THEN ID=1 ELSE ID=0
2170 IF ID=1 AND (P AND &HF)=&HF THEN 2410
2180 IF ID=0 THEN 2210
2190 GOSUB 2210
2200 OPR$="["+OPR$+"]":RETURN
2210 ON (P AND &HF)+1 GOTO 2230,2240,2250,2260,2270,2290,2300,2400,2310,2350,240
0,2280,2380,2390,2400,2400
2220 ON (P AND &HF)+1 GOTO 2230,2240,2250,2260,2270,2290,2300,2400,2310,2350,240
0,2280,2380,2390,2400,2400
2230 OPR$=","+R$+"+":IF ID=1 THEN 2400 ELSE RETURN:' AUTO +1
2240 OPR$=","+R$+"++":RETURN:' AUTO +2
2250 OPR$=",-"+R$:IF ID=1 THEN 2400 ELSE RETURN:' AUTO -1
2260 OPR$=",--"+R$:RETURN:' AUTO -2
2270 OPR$=","+R$:RETURN:' 0 OFFSET
2280 OPR$="D,"+R$:RETURN:' D reg OFFSET
2290 OPR$="B,"+R$:RETURN:' B reg OFFSET
2300 OPR$="A,"+R$:RETURN:' A reg OFFSET
2310 A=PEEK(AD):AD=AD+1:IF A>127 THEN OPR$="-$":A=256-A ELSE OPR$="$":' 8bit
2320 OPR$=OPR$+FND$(A)+","+R$:RETURN
2330 A=P AND &H1F:IF A>15 THEN OPR$="-$":A=32-A ELSE OPR$="$":' 5 bit
2340 GOTO 2320
2350 A=PEEK(AD)*256+PEEK(AD+1):AD=AD+2:' 16 bit
2360 IF A>32767 THEN OPR$="-$":A=65536!-A ELSE OPR$="$"
2370 OPR$=OPR$+FNA$(A)+","+R$:RETURN
2380 GOSUB 2040:OPR$=OPR$+",PCR":RETURN:' PCrel 8bit
2390 GOSUB 2060:OPR$=OPR$+",PCR":RETURN:' PCrel 16bit
2400 OPC$="?":OPR$="":AD=SAD+1:RETURN
2410 OPR$="[$"+FNA$(PEEK(AD)*256+PEEK(AD+1))+"]":AD=AD+2:RETURN
3000 ' data read routine
3010 DIM REG$(15):RESTORE3250:FOR I=0 TO 15:READ REG$(I):NEXT
3020 DIM PREG$(7):RESTORE3270:FOR I=0 TO 7:READ PREG$(I):NEXT
3030 DIM IREG$(3):RESTORE3290:FOR I=0 TO 3:READ IREG$(I):NEXT
3040 DIM N0$(15):RESTORE3120:FOR I=0 TO 15:READ N0$(I):NEXT
3050 DIM N1$(15):RESTORE3140:FOR I=0 TO 15:READ N1$(I):NEXT
3060 DIM N2$(15):RESTORE3160:FOR I=0 TO 15:READ N2$(I):NEXT
3070 DIM N3$(15):RESTORE3180:FOR I=0 TO 15:READ N3$(I):NEXT
3080 DIM N8$(15):RESTORE3200:FOR I=0 TO 15:READ N8$(I):NEXT
3090 DIM NC$(15):RESTORE3230:FOR I=0 TO 15:READ NC$(I):NEXT
3100 RETURN
3110 ' data of MSN=0,4..7
3120 DATA NEG,?,?,COM,LSR,?,ROR,ASR,ASL,ROL,DEC,?,INC,TST,JMP,CLR
3130 ' data.of MSN=1
3140 DATA "","",NOP,SYNC,?,?,LBRA,LBSR,?,DAA,ORCC,?,ANDCC,SEX,EXG,TFR
3150 ' data of MSN=2
3160 DATA BRA,BRN,BHI,BLS,BCC,BCS,BNE,BEQ,BVC,BVS,BPL,BMI,BGE,BLT,BGT,BLE
3170 ' data of MSN=3
3180 DATA LEAX,LEAY,LEAS,LEAU,PSHS,PULS,PSHU,PULU,?,RTS,ABX,RTI,CWAI,MUL,?,SWI
3190 ' data of MSN=8..B
3200 DATA SUBA,CMPA,SBCA,SUBD,ANDA,BITA,LDA,STA
3210 DATA EORA,ADCA,ORA,ADDA,CMPX,JSR,LDX,STX
3220 ' data of MSN=C..F
3230 DATA SUBB,CMPB,SBCB,ADDD,ANDB,BITB,LDB,STB
3240 DATA EORB,ADCB,ORB,ADDB,LDD,STD,LDU,STU
3250 ' data of reg
3260 DATA D,X,Y,U,S,PC,**,**,A,B,CC,DP,**,**,**,**
3270 ' data of PREG
3280 DATA CC,A,B,DP,X,Y,S/U,PC
3290 ' data if IREG
3300 DATA X,Y,U,S
```

逆アセンブラ

これは，メモリ上に格納されているマシンコードを，アセンブリ言語へ逆翻訳するプログラムです。これはＢＡＳＩＣで組んであるので，マシン語プログラムが格納されているところはＣＬＥＡＲ文で使用させないようにするなどの処理を行ってから実行させてください。

このプログラムでは全体をとおして

Ｙレジスタ＝カーソル位置

Ｂレジスタ＝各行での左からのニブルでの位置

Ｕレジスタ＝プリンタフラグ

とレジスタを割りあててあります。

〔図付録−9　ＭＩＴ７ソースリスト〕

```
  1                                     ***********************
  2                                     *
  3                                     * Machine code
  4                                     * Input
  5                                     * Tool      (MIT7)
  6                                     * seven
  7                                     *
  8                                     * PROGRAMED BY
  9                                     *    H.NAKAMURA
 10                                     *
 11                                     *  POSITION INDEPENDENT
 12                                     *
 13                                     * coded 84/03/01
 14                                     *
 15                                     ***********************
 16                                     *
 17                                     * COPYRIGHT (C) 1984
 18                                     *        BY
 19                                     *    H.NAKAMURA
 20                                     *
 21                                     ***********************
 22
 23  1000                                       ORG     $1000      START OF PROGRAM
 24
 25                       1000  START   EQU     *
 26
 27  1000 CE   0000                     LDU     #0         PRINTER FLAG OFF
 28  1003 86   0C                       LDA     #$0C'      CLEAR SCREEN
 29  1005 17   02CE                     LBSR    CHROUT
 30  1008 17   036B             MAIN    LBSR    CURSOR     CURSOR ON
 31  100B CC    1800                     LDD     #24*$100   LOCATE 0,24
 32  100E 17   0344                     LBSR    LOCAT2
 33  1011 17   0292                     LBSR    CRLF       SCROLL UP 1 LINE
 34  1014 30   8D 03FD                  LEAX    MESADR,PCR 'ADDRESS $='
 35  1018 17   032D                     LBSR    PRINT
 36  101B 17   01ED                     LBSR    GETADR     GET ADDR. IN Dreg
 37  101E 1F   01                       TFR     D,X
 38  1020 34   10                       PSHS    X          SCREEN TOP ADDRESS
 39  1022 4F                            CLRA               posX
 40  1023 5F                    SCR1    CLRB               posY
 41  1024 17   032E                     LBSR    LOCAT2     LOCATE Areg,Breg
 42  1027 17   0208                     LBSR    DUMP       DUMP 1 LINE
 43  102A 30   88 10                    LEAX    16,X       NEXT LINE TOP ADDR.
 44  102D 4C                            INCA
 45  102E 81   19                       CMPA    #25        END OF LINE ?
 46  1030 26   F1                       BNE     SCR1
 47  1032 35   10                       PULS    X
 48
 49  1034 10BE 0006             EDIT    LDY     #6         LOCATE 6,0
 50  1038 17   031E                     LBSR    LOCATE
 51  103B 5F                            CLRB               Breg=OFFSET
 52  103C 17   02C9             LOOP    LBSR    CHRIN      KEY INPUT 1 CHR
 53  103F 81   09                       CMPA    #$09       TAB Teach Addr. Base
 54 >1041 1027 FFC3                     LBEQ    MAIN
 55  1045 81   1B                       CMPA    #$1B       ESC ESCape to system
 56  1047 1027 01B6                     LBEQ    END
 57  104B 81   11                       CMPA    #$11       DUP DUmp to Printer
 58  104D 1027 0143                     LBEQ    PDUMP
 59  1051 81   08                       CMPA    #$08       BS
 60  1053 1027 00FB                     LBEQ    LEFT
 61  1057 81   0D                       CMPA    #$0D       CR AS LEFT
 62  1059 1027 00F5                     LBEQ    LEFT
 63  105D 81   2E                       CMPA    #'.        '. AS RIGHT
 64  105F 1027 00B6                     LBEQ    RIGHT
 65  1063 81   1C                       CMPA    #$1C       ->
 66  1065 1027 00B0                     LBEQ    RIGHT
 67  1069 81   1D                       CMPA    #$1D       <-
 68  106B 1027 00E3                     LBEQ    LEFT
 69  106F 81   1E                       CMPA    #$1E       ^
 70  1071 1027 00F6                     LBEQ    UP
 71  1075 81   1F                       CMPA    #$1F       V
 72  1077 1027 00BB                     LBEQ    DOWN
 73  107B 34   04                       PSHS    B
 74  107D 5F                            CLRB
 75  107E 81   2A                       CMPA    #'*        * AS A
 76  1080 27   25                       BEQ     H10
 77  1082 81   2F                       CMPA    #'/        / AS B
 78  1084 27   20                       BEQ     H11
 79  1086 81   2B                       CMPA    #'+        + AS C
 80  1088 27   1B                       BEQ     H12
 81  108A 81   2D                       CMPA    #'-        - AS D
 82  108C 27   16                       BEQ     H13
 83  108E 81   3D                       CMPA    #'=        = AS E
 84  1090 27   11                       BEQ     H14
 85  1092 81   2C                       CMPA    #',        , AS F
 86  1094 27   0C                       BEQ     H15
 87  1096 81   61                       CMPA    #'a        a-z -> A-Z
 88  1098 25   11                       BCS     HC
 89  109A 81   7A                       CMPA    #'z
```

```
   90   109C 22   0D                BHI    HC
   91   109E 88   20                EORA   #$20
   92   10A0 20   09                BRA    HC
   93   10A2 5C           H15       INCB
   94   10A3 5C           H14       INCB
   95   10A4 5C           H13       INCB
   96   10A5 5C           H12       INCB
   97   10A6 5C           H11       INCB
   98   10A7 CB   41      H10       ADDB   #'A
   99   10A9 1F   98                TFR    B,A       Areg= INKEY CODE
  100   10AB 17   0206    HC        LBSR   TSTHEX    TEST HEXADECIMAL
  101   10AE 1F   98                TFR    B,A       Areg= 0 - F
  102   10B0 35   04                PULS   B
  103   10B2 25   88                BCS    LOOP      IF NOT HEX THEN JUMP
  104
  105   10B4 34   16                PSHS   A,B,X
  106   10B6 54                     LSRB             OFFSET/2
  107   10B7 3A                     ABX              GET ADDRESS
  108   10B8 25   17                BCS    LOW       IF LSN(least significant nib
ble)
  109   10BA E6   E4                LDB    ,S        GET KEYIN NUM.
  110   10BC 58                     LSLB             KEYIN*16
  111   10BD 58                     LSLB
  112   10BE 58                     LSLB
  113   10BF 58                     LSLB
  114   10C0 E7   E4                STB    ,S
  115   10C2 A6   84                LDA    ,X        GET MEMORY DATA
  116   10C4 84   0F                ANDA   #$0F      MASK MSN
  117   10C6 AB   E0                ADDA   ,S+       SET KEYIN
  118   10C8 A7   84                STA    ,X        SET TO MEMORY
  119   10CA A6   84                LDA    ,X        GET MEMORY DATA
  120   10CC 17   01B7              LBSR   HEXOUT    OUTPUT MEM DATA
  121   10CF 20   14                BRA    CSOUT
  122
  123   10D1 A6   84      LOW       LDA    ,X        GET MEM DATA
  124   10D3 84   F0                ANDA   #$F0      MASK LSN
  125   10D5 AB   E0                ADDA   ,S+       SET KEYIN
  126   10D7 A7   84                STA    ,X        SET TO MEMORY
  127   10D9 A6   84                LDA    ,X        GET MEM DATA
  128   10DB 31   3F                LEAY   -1,Y      posX=posX-1
  129   10DD 17   0279              LBSR   LOCATE
  130   10E0 17   01A3              LBSR   HEXOUT    OUTPUT MEM DATA
  131   10E3 31   21                LEAY   1,Y       posX=posX+1
  132
  133   10E5 1F   20      CSOUT     TFR    Y,D       CHECKSUM_DISPLAY
  134   10E7 C6   37                LDB    #55       LOCATE 55,posY
  135   10E9 17   0269              LBSR   LOCAT2
  136   10EC 5F                     CLRB
  137   10ED 4F                     CLRA
  138   10EE AE   61                LDX    1,S       GET LINE TOP ADDR
  139   10F0 AB   85      CS2       ADDA   B,X       16BYTE ADD
  140   10F2 5C                     INCB
  141   10F3 C1   10                CMPB   #16
  142   10F5 26   F9                BNE    CS2
  143   10F7 17   018C              LBSR   HEXOUT    CHECKSUM OUTPUT
  144
  145   10FA 1F   20                TFR    Y,D       OUTPUT ASCII CHR
  146   10FC E6   E4                LDB    ,S        GET OFFSET
  147   10FE 54                     LSRB             OFFSET/2
  148   10FF 3A                     ABX              GET ADDRESS
  149   1100 CB   3B                ADDB   #59       GET posX
  150   1102 17   0250              LBSR   LOCAT2
  151   1105 A6   84                LDA    ,X        CTRL KEY ->'.'
  152   1107 81   7F                CMPA   #$7F
  153   1109 27   04                BEQ    CS3
  154   110B 81   20                CMPA   #'        SPACE
  155   110D 24   02                BCC    CS4
  156   110F 86   2E      CS3       LDA    #'.
  157   1111 17   01C2    CS4       LBSR   CHROUT    OUTPUT CHR
  158   1114 17   0242              LBSR   LOCATE
  159   1117 35   14                PULS   B,X
  160
  161   1119 5C           RIGHT     INCB             RIGHT MOVE
  162   111A C1   20                CMPB   #32       OVER ?
  163   111C 27   0E                BEQ    RIGHT2
  164   111E C5   01                BITB   #$01
  165   1120 26   02                BNE    RIGHT1    IF LSN(NEW)
  166   1122 31   21                LEAY   1,Y       IF MSN(NEW)
  167   1124 31   21      RIGHT1    LEAY   1,Y
  168   1126 17   0230              LBSR   LOCATE
  169   1129 16   FF10              LBRA   LOOP
  170   112C 1F   20      RIGHT2    TFR    Y,D       LEFT TOP & DOWM
  171   112E C6   06                LDB    #6
  172   1130 1F   02                TFR    D,Y
  173   1132 5F                     CLRB
  174
  175   1133 30   88 10   DOWN      LEAX   16,X      NEXT LINE
  176   1136 31   A9 0100           LEAY   $100,Y    posY=posY+1
  177   113A 108C 1900              CMPY   #$100*25  posY OVER 24 ?
  178   113E 24   06                BCC    DOWN2
```

```
179   1140 17    0216      DOWN1   LBSR   LOCATE
180   1143 16    FEF6              LBRA   LOOP
181   1146 17    015D      DOWN2   LBSR   CRLF         UP SCROOL
182   1149 17    00E6              LBSR   DUMP         DUMP 16BYTE
183   114C 31    A9 FF00           LEAY   -$100,Y      posY=24
184   1150 20    EE                BRA    DOWN1
185
186   1152 5A              LEFT    DECB                LEFT MOVE
187   1153 2B    0E                BMI    LEFT2        OVER ?
188   1155 C5    01                BITB   #$01
189   1157 27    02                BEQ    LEFT1
190   1159 31    3F                LEAY   -1,Y         IF LSN(NEW)
191   115B 31    3F        LEFT1   LEAY   -1,Y
192   115D 17    01F9              LBSR   LOCATE
193   1160 16    FED9              LBRA   LOOP
194   1163 1F    20        LEFT2   TFR    Y,D          LINE END & UP LINE
195   1165 C6    34                LDB    #52          posX=52
196   1167 1F    02                TFR    D,Y
197   1169 C6    1F                LDB    #31          OFFSET=31
198
199   116B 30    10        UP      LEAX   -16,X        UP LINE
200   116D 31    A9 FF00           LEAY   -$100,Y      posY=posY-1
201   1171 108C  0000              CMPY   #0
202   1175 2D    06                BLT    UP2          OVER ?
203   1177 17    01DF      UP1     LBSR   LOCATE
204   117A 16    FEBF              LBRA   LOOP
205   117D 31    A9 0100   UP2     LEAY   $100,Y       DUWN SCROOL
206   1181 17    0201              LBSR   OFFCUR       CORSOR OFF
207   1184 17    020E              LBSR   DWNSC        DUWN SCROOL
208   1187 86    0B                LDA    #$0B         HOME
209   1189 17    014A              LBSR   CHROUT
210   118C 17    00A3              LBSR   DUMP         DUMP 16BYTE
211   118F 17    01E4              LBSR   CURSOR
212   1192 20    E3                BRA    UP1
213
214   1194 34    76        PDUMP   PSHS   D,X,Y,U      PRINTER OUTPUT
215   1196 CC    1800              LDD    #24*$100     LOCATE 0,24
216   1199 17    01B9              LBSR   LOCAT2
217   119C 17    0107              LBSR   CRLF         UP SCROOL
218   119F B6    FD02              LDA    $FD02        CHECK PRINTER ERROR
219   11A2 85    02                BITA   #$02
220   11A4 27    4B                BEQ    PDUMP2       IF ERROR
221   11A6 86    04                LDA    #4
222   11A8 108E  0000      PDUMPA  LDY    #0           WAIT UNTIL READY
223   11AC F6    FD02      PDUMPB  LDB    $FD02
224   11AF 54                      LSRB
225   11B0 24    09                BCC    PDUMPC
226   11B2 31    3F                LEAY   -1,Y
227   11B4 26    F6                BNE    PDUMPB
228   11B6 4A                      DECA
229   11B7 26    EF                BNE    PDUMPA
230   11B9 20    36                BRA    PDUMP2
231   11BB 30    8D 0250   PDUMPC  LEAX   MESSTR,PCR GET START ADDR IN Xreg
232   11BF 17    0186              LBSR   PRINT
233  >11C2 17    0046              LBSR   GETADR
234   11C5 1F    02                TFR    D,Y
235   11C7 17    00DC              LBSR   CRLF        GET END ADDR IN STACKTOP
236   11CA 30    8D 0252           LEAX   MESEND,PCR
237   11CE 17    0177              LBSR   PRINT
238   11D1 30    8D 0240           LEAX   MESADR,PCR
239   11D5 17    0170              LBSR   PRINT
240  >11D8 17    0030              LBSR   GETADR
241   11DB 34    06                PSHS   D
242   11DD 1F    21                TFR    Y,X
243   11DF CE    0001              LDU    #1           PRINTER FLAG ON
244   11E2 AC    E4        PDUMP1  CMPX   ,S           END CHECK
245   11E4 22    14                BHI    PDUMP3
246  >11E6 17    0049              LBSR   DUMP         DUMP 1 LINE
247   11E9 17    00BA              LBSR   CRLF
248   11EC 30    88 10             LEAX   16,X
249   11EF 20    F1                BRA    PDUMP1
250   11F1 30    8D 0232   PDUMP2  LEAX   MESERR,PCR
251   11F5 17    0150              LBSR   PRINT
252   11F8 20    02                BRA    PDUMP4
253   11FA 35    06        PDUMP3  PULS   D
254   11FC 35    76        PDUMP4  PULS   D,X,Y,U
255   11FE 16    FE07              LBRA   MAIN
256
257   1201 CC    1800      END     LDD    #24*$100     EXIT TO SYSTEM
258   1204 17    014E              LBSR   LOCAT2       LOCATE 0,24
259   1207 17    009C              LBSR   CRLF         UP SCROOL
260   120A 39                      RTS
261
262                        **************
263                        *
264                        * SUBROUTINES
265                        *
266                        **************
267   120B CC    0000      GETADR  LDD    #0           GET ADDRESS IN Dreg
268   120E 34    06                PSHS   D            ADDR.
```

```
269    1210 4F         GA1     CLRA
270    1211 34    06           PSHS    D          KEYIN
271    1213 EC    62           LDD     2,S        GET OLD ADDR
272    1215 58                 ASLB               ADDR*16
273    1216 49                 ROLA
274    1217 58                 ASLB
275    1218 49                 ROLA
276    1219 58                 ASLB
277    121A 49                 ROLA
278    121B 58                 ASLB
279    121C 49                 ROLA
280    121D E3    E1           ADDD    ,S++       ADD KEYIN
281    121F ED    E4           STD     ,S
282    1221 17    00E4  GA2    LBSR    CHRIN      KEY INPUT
283    1224 17    00AF         LBSR    CHROUT     ECHO BACK
284    1227 17    00BA         LBSR    TSTHEX     GET HEXNUM
285    122A 24    E4           BCC     GA1        IF HEX
286    122C 81    0D           CMPA    #$0D
287    122E 26    F1           BNE     GA2
288    1230 35    86           PULS    D,PC
289
290    1232 34    16    DUMP   PSHS    D,X        DUMP 1 LINE
291    1234 1F    10           TFR     X,D        ADDRESS OUTPUT
292   >1236 17    004D         LBSR    HEXOUT
293    1239 1F    98           TFR     B,A
294   >123B 17    0048         LBSR    HEXOUT
295    123E 86    3A           LDA     #':
296    1240 17    0093         LBSR    CHROUT
297    1243 34    10           PSHS    X
298    1245 4F                 CLRA
299    1246 34    02           PSHS    A
300    1248 C6    10           LDB     #16        16BYTE OUTPUT
301   >124A 17    0050  DU1    LBSR    SPOUT
302    124D A6    84           LDA     ,X
303   >124F 17    0034         LBSR    HEXOUT
304    1252 A6    80           LDA     ,X+
305    1254 AB    E4           ADDA    ,S         FOR CHECKSUM
306    1256 A7    E4           STA     ,S
307    1258 5A                 DECB
308    1259 26    EF           BNE     DU1
309   >125B 17    003F         LBSR    SPOUT
310    125E 86    3A           LDA     #':
311   >1260 17    0073         LBSR    CHROUT
312    1263 35    02           PULS    A          CHECKSUM OUTPUT
313   >1265 17    001E         LBSR    HEXOUT
314   >1268 17    0032         LBSR    SPOUT
315   >126B 17    002F         LBSR    SPOUT
316    126E 35    10           PULS    X
317    1270 C6    10           LDB     #16        ASCII DUMP
318    1272 A6    80    DU2    LDA     ,X+
319    1274 81    7F           CMPA    #$7F
320    1276 27    04           BEQ     DU3
321    1278 81    20           CMPA    #'         SPACE
322    127A 24    02           BCC     DU4
323    127C 86    2E    DU3    LDA     #'.
324   >127E 17    0055  DU4    LBSR    CHROUT
325    1281 5A                 DECB
326    1282 26    EE           BNE     DU2
327    1284 35    96           PULS    D,X,PC
328
329    1286 34    02    HEXOUT PSHS    A          OUTPUT Areg IN HEX
330    1288 44                 LSRA
331    1289 44                 LSRA
332    128A 44                 LSRA
333    128B 44                 LSRA
334   >128C 17    0004         LBSR    HEX1       OUTPUT MSN
335    128F 35    02           PULS    A
336    1291 84    0F           ANDA    #$0F
337    1293 81    0A    HEX1   CMPA    #10        OUTPUT LSN
338    1295 25    02           BCS     HEX2
339    1297 8B    07           ADDA    #'A-'9-1
340    1299 8B    30    HEX2   ADDA    #'0
341    129B 20    39           BRA     CHROUT
342
343    129D 34    02    SPOUT  PSHS    A          OUTPUT SPACE
344    129F 86    20           LDA     #'         SPACE
345   >12A1 17    0032         LBSR    CHROUT
346    12A4 35    82           PULS    A,PC
347
348    12A6 34    02    CRLF   PSHS    A          OUTPUT CR & LF
349    12A8 86    0D           LDA     #$0D
350   >12AA 17    0029         LBSR    CHROUT
351    12AD 86    0A           LDA     #$0A
352   >12AF 17    0024         LBSR    CHROUT
353    12B2 35    82           PULS    A,PC
354
355    12B4 34    02    TSTHEX PSHS    A          CHECK HEX
356    12B6 81    30           CMPA    #'0
357    12B8 25    18           BCS     TS2
358    12BA 81    39           CMPA    #'9
```

```
359    12BC 22    04                        BHI     TS1
360    12BE 80    30                        SUBA    #'0
361    12C0 20    0A                        BRA     TS3
362    12C2 81    41            TS1         CMPA    #'A
363    12C4 25    0C                        BCS     TS2
364    12C6 81    46                        CMPA    #'F
365    12C8 22    08                        BHI     TS2
366    12CA 80    37                        SUBA    #'A-10
367    12CC 1F    89            TS3         TFR     A,B
368    12CE 1C    FE                        ANDCC   #$FE        IF HEX THEN C=0
369    12D0 35    82                        PULS    A,PC
370    12D2 1A    01            TS2         ORCC    #$01        IF NOT HEX THEN C=1
371    12D4 35    82                        PULS    A,PC
372
373                             ************
374                             *
375                             * I/O
376                             * SUBROUTINES
377                             *
378                             ************
379    12D6 34    06            CHROUT  PSHS    D           OUTPUT CHR BY 'PUTC'
380    12D8 1183  0000                  CMPU    #0          IF PRINTER
381    12DC 26    11                    BNE     POUT
382    12DE 17    010F                  LBSR    SSTOP
383    12E1 B7    FC84                  STA     $FC84       SET CHR
384    12E4 CC    0301                  LDD     #$0301
385    12E7 FD    FC82                  STD     $FC82
386    12EA 17    0116                  LBSR    SRUN
387    12ED 35    86                    PULS    D,PC
388
389    12EF F6    FD02          POUT    LDB     $FD02       PRINTER OUTPUT
390    12F2 C5    02                    BITB    #$02
391    12F4 27    10                    BEQ     POUT1
392    12F6 54                          LSRB
393    12F7 25    F6                    BCS     POUT
394    12F9 B7    FD01                  STA     $FD01       SET CHR
395    12FC C6    00                    LDB     #0          OUTPUT STROBE
396    12FE F7    FD00                  STB     $FD00
397    1301 C6    40                    LDB     #$40
398    1303 F7    FD00                  STB     $FD00
399    1306 35    86            POUT1   PULS    D,PC
400
401    1308 34    04            CHRIN   PSHS    B
402    130A CC    0500          CHRIN1  LDD     #$500       WAIT FOR CURSOR BLINK
403    130D 83    0001          CHRIN2  SUBD    #1
404    1310 26    FB                    BNE     CHRIN2
405    1312 17    00DB                  LBSR    SSTOP
406    1315 CC    2900                  LDD     #$2900      INKEY COMMAND
407    1318 FD    FC82                  STD     $FC82
408    131B 17    00E5                  LBSR    SRUN
409    131E 17    00CF                  LBSR    SSTOP
410    1321 B6    FC80                  LDA     $FC80       SET READY REQUEST
411    1324 8A    80                    ORA     #$80
412    1326 B7    FC80                  STA     $FC80
413    1329 FC    FC83                  LDD     $FC83
414    132C 17    00D4                  LBSR    SRUN
415    132F 5D                          TSTB                KEYIN ?
416    1330 27    D8                    BEQ     CHRIN1      NO THEN AGAIN
417    1332 34    02                    PSHS    A
418    1334 C6    81                    LDB     #$81        SOUND OUTPUT
419    1336 F7    FD03                  STB     $FD03
420    1339 CC    0400                  LDD     #$0400
421    133C 83    0001          CHRIN3  SUBD    #1
422    133F 26    FB                    BNE     CHRIN3
423    1341 C6    00                    LDB     #$00
424    1343 F7    FD03                  STB     $FD03
425    1346 35    86                    PULS    A,B,PC
426
427    1348 34    12            PRINT   PSHS    A,X         BRINTOUT from X to (0)
428    134A A6    80            PRINT2  LDA     ,X+
429    134C 27    05                    BEQ     PRINT1
430   >134E 17    FF85                  LBSR    CHROUT
431    1351 20    F7                    BRA     PRINT2
432    1353 35    92            PRINT1  PULS    A,X,PC
433
434    1355 34    06            LOCAT2  PSHS    D           LOCATE Breg,Areg
435    1357 20    04                    BRA     LOCAT3
436    1359 34    06            LOCATE  PSHS    D           LOCATE Yreg
437    135B 1F    20                    TFR     Y,D
438    135D 17    0090          LOCAT3  LBSR    SSTOP
439    1360 B7    FC86                  STA     $FC86       posY
440    1363 F7    FC85                  STB     $FC85       posY
441    1366 CC    0303                  LDD     #$0303
442    1369 FD    FC82                  STD     $FC82
443    136C 86    12                    LDA     #$12        order LOCATE
444    136E B7    FC84                  STA     $FC84
445    1371 17    008F                  LBSR    SRUN
446    1374 35    86                    PULS    D,PC
447
448    1376 34    06            CURSOR  PSHS    D           corsor display
```

```
449    1378 8D    76                    BSR    SSTOP
450    137A CC    0C1F                  LDD    #$0C*256+%011111
451    137D FD    FC82                  STD    $FC82
452    1380 17    0080                  LBSR   SRUN
453    1383 35    86                    PULS   D,PC
454
455    1385 34    06          OFFCUR    PSHS   D           cursor non-display
456   >1387 17    0066                  LBSR   SSTOP
457    138A CC    0C1E                  LDD    #$0C*256+%011110
458    138D FD    FC82                  STD    $FC82
459   >1390 17    0070                  LBSR   SRUN
460    1393 35    86                    PULS   D,PC
461
462    1395 34    56          DWNSC     PSHS   D,X,U       dowm scroll
463    1397 CC    1800                  LDD    #24*$100    bottom line erase
464   >139A 17    FFB8                  LBSR   LOCAT2
465    139D 86    20                    LDA    #'          SPACE
466    139F C6    4F                    LDB    #79
467    13A1 17    FF32        DWNSC0    LBSR   CHROUT
468    13A4 5A                          DECB
469    13A5 26    FA                    BNE    DWNSC0
470    13A7 8D    47                    BSR    SSTOP
471    13A9 30    8D 0010               LEAX   SUBCOM,PCR set program in shared-RAM
472    13AD CE    FC80                  LDU    #$FC80
473    13B0 C6    33                    LDB    #_SUBCM-SUBCOM
474    13B2 A6    80          DWNSC1    LDA    ,X+
475    13B4 A7    C0                    STA    ,U+
476    13B6 5A                          DECB
477    13B7 26    F9                    BNE    DWNSC1
478    13B9 8D    48                    BSR    SRUN
479    13BB 35    D6                    PULS   D,X,U,PC
480    13BD 00 00 3F          SUBCOM    FCB    0,0,$3F     shared-RAM program
481    13C0 59 41 4D 41                 FCC    'YAMAUCHI
       13C4 55 43 48 49
482    13C8 93                          FCB    $93
483    13C9 D3BF                        FDB    $D3BF
484    13CB 90                          FCB    $90
485    13CC FC    D01F                  LDD    $D01F       set VRAM offset
486    13CF B3    D03B                  SUBD   $D03B
487    13D2 7D    D409                  TST    $D409       VRAM flag on
488    13D5 FD    D01F                  STD    $D01F
489    13D8 FD    D40E                  STD    $D40E
490    13DB B7    D409                  STA    $D409       VRAM flag off
491    13DE 34    10                    PSHS   X
492    13E0 8E    CFA0                  LDX    #$C000+80*50 move console CHRs
493    13E3 A6    82          SUBCML    LDA    ,-X
494    13E5 A7    88 50                 STA    80,X
495    13E8 8C    C000                  CMPX   #$C000
496    13EB 26    F6                    BNE    SUBCML
497    13ED 35    10                    PULS   X
498    13EF 39                          RTS
499                    13F0   _SUBCM    EQU    *           end of subCPUprogram
500
501    13F0 34    02          SSTOP     PSHS   A           Stop subCPU
502    13F2 B6    FD05        SSTOP1    LDA    $FD05
503    13F5 2B    FB                    BMI    SSTOP1
504    13F7 86    80                    LDA    #$80
505    13F9 B7    FD05                  STA    $FD05
506    13FC B6    FD05        SSTOP2    LDA    $FD05
507    13FF 2A    FB                    BPL    SSTOP2
508    1401 35    82                    PULS   A,PC
509
510    1403 34    02          SRUN      PSHS   A           restart subCPU
511    1405 7F    FD05                  CLR    $FD05
512    1408 86    18                    LDA    #24
513    140A 4A                SRUN1     DECA
514    140B 26    FD                    BNE    SRUN1
515    140D 35    82                    PULS   A,PC
516
517    140F 53 54 41 52       MESSTR    FCC    'START '    messages
       1413 54 20
518    1415 41 44 44 52       MESADR    FCC    'ADDRESS =$'
       1419 45 53 53 20
       141D 3D 24
519    141F 00                          FCB    0
520    1420 45 4E 44 20       MESEND    FCC    'END
       1424 20 20
521    1426 00                          FCB    0
522    1427 50 52 49 4E       MESERR    FCC    'PRINTER ERROR !!
       142B 54 45 52 20
       142F 45 52 52 4F
       1433 52 20 21 21
523    1437 00                          FCB    0
524                                     END    START

0 ERROR(S) DETECTED

SYMBOL TABLE:
```

```
CHRIN   1308   CHRIN1  130A   CHRIN2  130D   CHRIN3  133C   CHROUT  12D6
CRLF    12A6   CS2     10F0   CS3     110F   CS4     1111   CSOUT   10E5
CURSOR  1376   DOWN    1133   DOWN1   1140   DOWN2   1146   DU1     124A
DU2     1272   DU3     127C   DU4     127E   DUMP    1232   DWNSC   1395
DWNSC0  13A1   DWNSC1  13B2   EDIT    1034   END     1201   GA1     1210
GA2     1221   GETADR  120B   H10     10A7   H11     10A6   H12     10A5
H13     10A4   H14     10A3   H15     10A2   HC      10AB   HEX1    1293
HEX2    1299   HEXOUT  1286   LEFT    1152   LEFT1   115B   LEFT2   1163
LOCAT2  1355   LOCAT3  135D   LOCATE  1359   LOOP    103C   LOW     10D1
MAIN    1008   MESADR  1415   MESEND  1420   MESERR  1427   MESSTR  140F
OFFCUR  1385   PDUMP   1194   PDUMP1  11E2   PDUMP2  11F1   PDUMP3  11FA
PDUMP4  11FC   PDUMPA  11A8   PDUMPB  11AC   PDUMPC  11BB   POUT    12EF
POUT1   1306   PRINT   1348   PRINT1  1353   PRINT2  134A   RIGHT   1119
RIGHT1  1124   RIGHT2  112C   SCR1    1023   SPOUT   129D   SRUN    1403
SRUN1   140A   SSTOP   13F0   SSTOP1  13F2   SSTOP2  13FC   START   1000
SUBCML  13E3   SUBCOM  13BD   TS1     12C2   TS2     12D2   TS3     12CC
TSTHEX  12B4   UP      116B   UP1     1177   UP2     117D   _SUBCM  13F0
```

このＭＩＴ７は２章でダンプリストおよび使用法を掲載したマシン語入力ツールの全ソースリストです。ここでは紙面の都合上，詳しい解説はしませんが，ソースリストには要所要所に注釈を付記しておきましたので，それを参照して解読してみるとよいでしょう。

参考文献

1．ＦＭ－７／NEW７／77　各マニュアル類，富士通

2．ＭＣ6809－ＭＣ6809Ｅ　マイクロプロセッサプログラミングマニュアル，日本モトローラ著，ＣＱ出版

3．コンピュータ用語辞典，アンソニーチャウダー他著，講談社

4．ＲＡＭ（1978年７月号，50～51ページ），広済堂出版

5．ＦＭ－７　Ｆ－ＢＡＳＩＣ解析マニュアル　フェーズⅠ　基礎編，拙著

6．ＦＭ－７　Ｆ－ＢＡＳＩＣ解析マニュアル　フェーズⅡ　探求編，箕原・菊地・南著

7．ＦＭ－７／NEW７／77　Ｆ－ＢＡＳＩＣ解析マニュアル　サブシステム編，菊地・箕原・南著

8．ＦＭ－７／NEW７／77　ユーティリティ・プログラム応用実例集，共著，監修　中村

9．ＦＭ－７・８　ＯＳ－９　Level Ⅰ　解析マニュアルⅠ　金井　隆著

5．～9．　秀和システムトレーディング株式会社